VŠB – Technická univerzita Ostrava
Fakulta elektrotechniky a informatiky
Katedra informatiky

Nástroj pro podporu "review" při vývoji systémů

Tool for System Development Items Review Support

2019 Bc. Josef Pohrom

VŠB - Technická univerzita Ostrava
Fakulta elektrotechniky a informatiky
Katedra informatiky

Zadání diplomové práce

Student: **Bc. Josef Pohrom**

Studijní program: N2647 Informační a komunikační technologie

Studijní obor: 2612T025 Informatika a výpočetní technika

Téma: Nástroj pro podporu "review" při vývoji systémů
Tool for System Development Items Review Support

Jazyk vypracování: čeština

Zásady pro vypracování:

Cílem diplomové je vytvoření nástroje pro podporu a záznam požadovaných review při vývoji systému za dodržení standardu Automotive SPICE. Prototypový nástroj bude obsahovat možnost podpory review otázkami, prázdné review formuláře pro záznam nespecifikovaného review, možnost přiřadit review kritéria pro různé pracovní produkty, soupis potřebných review a možnost napojení na plánovací vrstvu.

Práce bude obsahovat zejména:
1. Seznámení se s problematikou a analýzou případných dostupných nástrojů.
2. Architekturu výsledného nástroje.
3. Design a implementaci nástroje a ukázky použití.
4. Závěr, zhodnocení nástroje, případně porovnání s existujícími nástroji, vyhodnocení.

Seznam doporučené odborné literatury:

[1] John F. Sowa, Knowledge Representation: Logical, Philosophical, and Computational Foundations, Brooks Cole Publishing Co., Pacific Grove, CA, ©2000
[2] Alec Sharp, Patrick McDermott: Workflow Modeling: Tools for Process Improvement and Application Development, Artech House; 2 edition (October 31, 2008)
[3] Pfleeger, Shari Lawrence, and Joanne M. Atlee. 2009. Software Engineering: Theory and Practice: Prentice Hall, ISBN 0136061699
[4] Pressman, Roger S. 2010. Software Engineering : A Practitioner's Approach. 7th ed. New York: McGraw-Hill Higher Education, ISBN 9780073375977
[5] Sommerville, Ian. 2010. Software Engineering. 9th ed, International Computer Science Series. Harlow: Addison-Wesley, ISBN 978-0137035151
[6] Automotive SPICE® Process Reference and Assessment Model (PDF 1800KB) - RELEASE 3.1 - 01 November 2017, http://www.automotivespice.com/download/

Další literatura podle pokynů vedoucího diplomové práce.

Formální náležitosti a rozsah diplomové práce stanoví pokyny pro vypracování zveřejněné na webových stránkách fakulty.

Vedoucí diplomové práce: **Ing. Svatopluk Štolfa, Ph.D.**

Datum zadání: 01.09.2018

Datum odevzdání: 30.04.2019

doc. Ing. Jan Platoš, Ph.D.
vedoucí katedry

prof. Ing. Pavel Brandštetter, CSc.
děkan fakulty

Prohlašuji, že jsem tuto diplomovou práci vypracoval samostatně. Uvedl jsem všechny literární prameny a publikace, ze kterých jsem čerpal.

V Ostravě 22. dubna 2019 .....

Souhlasím se zveřejněním této diplomové práce dle požadavků čl. 26, odst. 9 Studijního a zkušebního řádu pro studium v magisterských programech VŠB-TU Ostrava.

V Ostravě 22. dubna 2019 .....

Rád bych poděkoval Ing. Svatopluku Štolfovi, Ph.D. za vedení této práce, za odbornou pomoc, rady a připomínky při tvorbě této diplomové práce. Dále bych chtěl poděkovat Bc. Michalu Přikrylovi za praktické rady při vývoji aplikace.

Abstrakt

Diplomová práce se zabývá problematikou provádění revizí při vývoji softwaru za dodržení standardu Automotive SPICE. V úvodní části jsou popsány procesy, které standard Automotive SPICE definuje a následně vysvětleno, dle jakých kriterii jsou tyto procesy hodnoceny v organizacích. Následně je popsáno jaké druhy revizí existují a je popsán proces u jednotlivých revizí. V předposlední části je proveden průzkum aplikací, které jsou na trhu a nabízejí funkcionalitu revidování artefaktů vývoje softwaru. Poslední část se zabývá vytvořením aplikace, která nabídne tuto funkcionalitu. Cílem této diplomové práce je vytvořit prototyp aplikace, která umožní vytvářet a spravovat revizní formuláře.

Klíčová slova: Automotive SPICE, revize, software, aplikace

Abstract

The diploma thesis deals with the issue of revisions in software development while complying with the Automotive SPICE standard. In the introductory part there are described the processes defined by Automotive SPICE and then explained by which criteria are these processes evaluated in organizations. Subsequently, what kinds of revisions exist and the process of individual revisions is described. The penultimate section explores the applications that are on the market and offers functionality to review software development artifacts. The last part deals with creating an application that offers this functionality. The aim of this thesis is to create a prototype application, which allows to create and manage revision forms.

Key Words: Automotive SPICE, review, software, application

Obsah

Seznam použitých zkratek a symbolů

HTML	–	Hyper Text Markup Language
MVC	–	Model View Controller
API	–	Application Programming Interface
JSON	–	JavaScript Object Notation
XML	–	Extensible Markup Language
REST	–	Representational State Transfer
AJAX	–	Asynchronous JavaScript and XML
JWT	–	JSON Web Token
SQL	–	Structured Query Language
SPICE	–	Software Process Improvement and Capability dEtermination
OSLC	–	Open Services for Lifecycle Colaboratio
ASP.NET	–	Active Server Pages .NET

Seznam obrázků

Seznam tabulek

Seznam výpisů zdrojového kódu

1 Úvod

Se zvětšujícími se rozsahy softwarových projektů, vzniká problém uhlídat, jestli jsou dodrženy veškeré náležitosti vývoje. Nesplnění těchto náležitostí má za důsledek zhoršení kvality odevzdaného softwaru. Tento problém dopomáhají řešit revize. Do revizí mohou být zahrnuty veškeré artefakty, které vznikly při vývoji softwaru, například uživatelské požadavky, dokumentace, testové případy a mnoho dalších. Tento proces přispívá k tomu, aby byla softwarová kvalita co nejlepší.

V první části této práce rozeberu standart Automotive SPICE, podle kterého se udržují firemní procesy v organizaci. Následně rozeberu jaké druhy revizí existují. Konkrétně to budou Software Peer review, Software Management review a Software audit. Popíši jakým způsobem se jednotlivé druhy revizi provádějí a jaké role jednotlivý účastníci obstarávají.

V druhé části práce popíšu svůj průzkum softwarů, které jsou na trhu. V této části jsem zkoumal softwary, které by mohly nabízet funkcionalitu, která by byla spjatá se softwarovými revizemi. Průzkumu jsem podrobil celkem čtyři softwary, kde hlavní funkcionalita, kterou jsem hledal, byla správa požadavků a jejich možnost následného revidování.

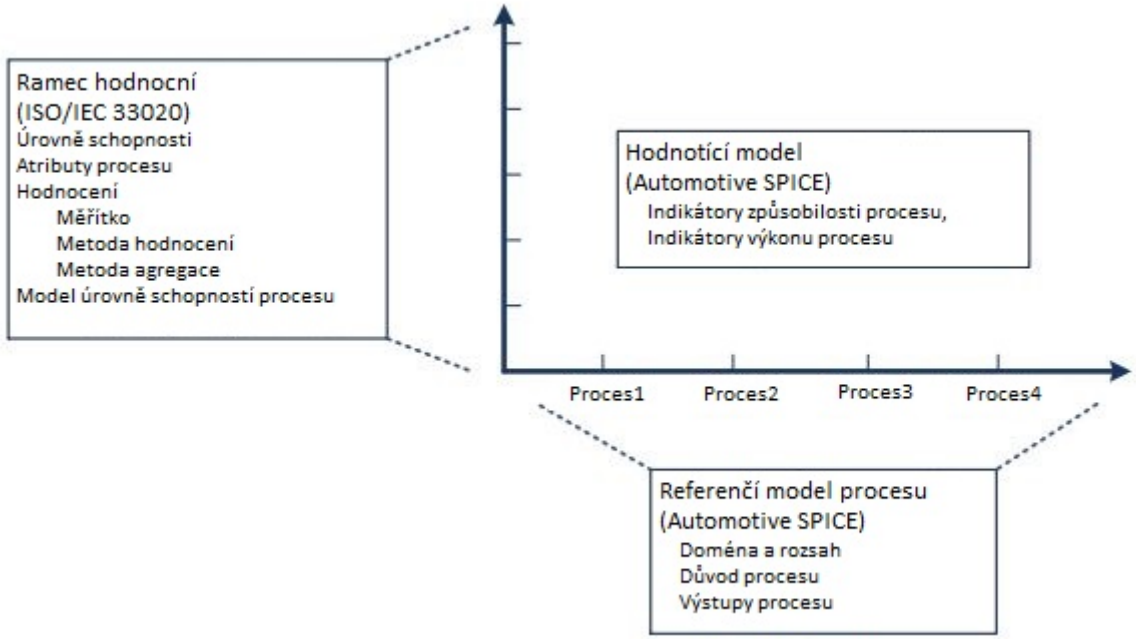
V poslední třetí části rozeberu svou aplikaci, která by měla dopomoct k řešení této problematiky. Popíši, jak je celá architektura aplikace navržena a jaké technologie využívám při její tvorbě.

2 Automotive SPICE

2.1 Hodnotící model

Automotive SPICE je standart, který stanovuje režim hodnocení procesů ve firmách a je založen na dvourozměrném frameworku. První dimenze je definována již stanovenými procesy ve vývojovém procesu firmy. Druhá dimenze je tvořena z úrovně schopností dodržet daný standart, který je dále rozdělen do procesních atributů. Tyto atributy zpřístupňují charakteristiky, které jsou měřitelné a tím definují procesu schopnost.[1]

Hodnotící model procesů vybírá procesy z referenčního procesního modelu (procesy , které jsou ve firmě již definovány), tím hodnotitelé jednotlivých procesů snadněji získávají objektivní důkazy, které jim umožní přiřadit hodnocení pro procesy, podle dimenze schopností. Vzniklý vztah mezi těmito dvěma dimenzemi je vidět na Obrázku 1.[1]



Obrázek 1: Hodnotící model procesů [1]

2.2 Referenční model procesu

Jednotlivé procesy podle standartu Automotive SPICE jsou rozděleny do kategorií. Tyto kategorie člení jednotlivé procesy podle toho, k jakému účelu tento proces slouží. Celý tento model je tedy rozdělen na tři hlavní části.

1. **Organizační procesy životního cyklu**
   Tato kategorie obsahuje veškeré procesy, které v organizaci vyvíjejí procesní, produktové a zdrojové prostředky (optimální rozdělení lidské síly na daný projekt). Tyto jednotlivé procesy poté pomáhají organizaci dosáhnout jejich obchodních cílů. Je rozdělená do tří podkategorii.[1]

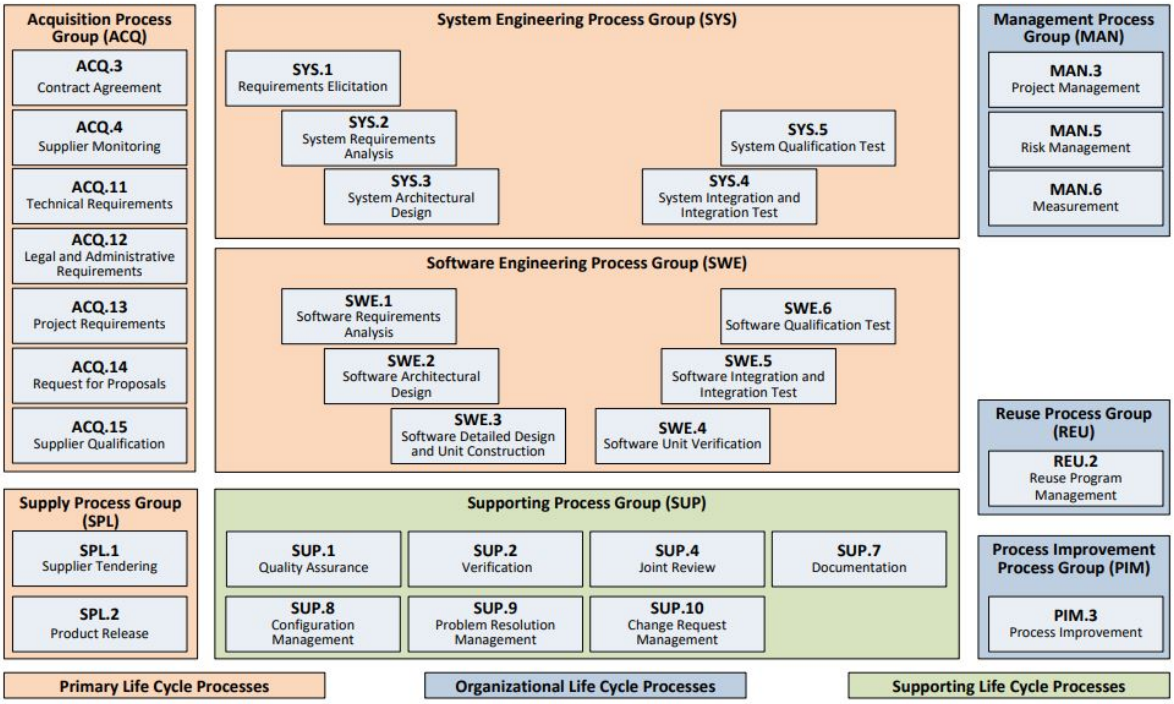
   - Skupina řídících procesů (Management process group)
   - Skupina procesů vylepšující stávající procesy (Process improvement process group)
   - Skupina procesů pro znovupoužití procesů (Reuse process group)

2. **Primární procesy životního cyklu**
   Tato kategorie obsahuje veškeré procesy, které mohou být využity zákazníkem, při domluvě požadavku na celý projekt, který bude dodavatel (organizace) zpracovávat. Toto platí i zpětně pro firmu, která při dodávání projektu nebo reagování na požadavky zákazníka v průběhu tvorby projektu. Tato kategorie zahrnuje specifikaci požadavků, designe, vývoj, integraci a testování. Tato kategorie je rozdělena do čtyř podkategorii.[1]

   - Skupina procesů nabytí akvizic (Acquisition process group)
   - Skupina procesů dodání produktů (Supply process group)
   - Skupina procesů systémového vývoje (System engineering process group)
   - Skupina procesů softwarového vývoje (Software Engineering process group)

3. **Podpůrné procesy životního cyklu**
   Tato kategorie se již dále nerozděluje do podkategorií a veškeré procesy zde obsažené mohou být použity kterýmkoliv z ostatních procesů obsaženém v tomto modelu a to v různých obdobích životního cyklu projektu. V další kapitole se pak budu konkrétně zabývat revidováním.[1]

Výše zmíněné kategorie a jejich členění do jednotlivých podkategorii, které pak dále obsahují již konkrétní procesy, jsou podrobně popsány ve Standartu Automotive SPICE. Konkrétní členění všech procesů lze vidět na Obrázku 2.



Obrázek 2: Referenční model Automotive SPICE [1]

2.3 Hodnocení procesu schopnosti v organizaci

Standart Automotive SPICE definuje úrovně vyspělosti procesů v organizaci, tyto úrovně vyspělosti v organizaci slouží k prokázání, že organizace je schopna zpracovat daný projekt. Těchto úrovní tento standart definuje celkem šest. Jednotlivé úrovně vždy splňují náležitosti předešlé úrovně (úroveň 2. splňuje veškeré náležitosti úrovně 1. a přidává řiditelnost daného procesu atd.). Popis jednotlivých úrovní procesů viz Tabulka 1. [1]

Úroveň 0: Nekompletní proces	Proces není nasazený (organizace nemá nijak definovaný postup práce) nebo nesplnila požadavky pro dosažení úrovně 1.
Úroveň 1: Performed process	Nasazený proces splňuje svůj účel a je specifický pro každý projekt (nelze znovu použít pro jiný projekt)
Úroveň 2: Řízený proces	Splňuje podmínky předchozí úrovně a zároveň je požíván řízeným způsobem(plánování, monitorování a úpravy) a veškeré podprocesy jsou náležitě zavedeny, řízeny a udržovány.
Úroveň 3: Zavedený proces	Splňuje náležitosti úrovně 2. a zároveň je proces definován, tak aby byl možno dosáhnout výsledků jednotlivých podprocesů.
Úroveň 4: Predikovaný proces	Splňuje veškeré náležitosti předchozí úrovně procesu a navíc pracuje v předvidatelných mezích, aby dosáhl výsledků procesu.
Úroveň 5: Inovativní proces	Splňuje veškeré náležitosti predikovaného procesu a navíc se dále vylepšuje tak, aby reagoval na organizační změny v organizaci.

Tabulka 1: Vyspělostní úrovně procesu dle standartu Automotive SPICE [1]

Aby organizace získala některou z úrovní vyspělosti, musí projít hodnocením jednotlivých procesů, které musí být splněny k dosažení dané úrovně. Toto hodnocení probíhá na základě dotazování se pracovníků hodnotiteli, kteří posuzují, jestli jsou dané náležitosti splněny. Splnění jednotlivých náležitostí je rozděleno do stupnice viz Tabulka 2.

N	Nesplnil	Je málo nebo žádné důkazy o splnění definované procesní vlastnosti v hodnoceném procesu.	0% až $\leq$ 15%
P	Částečně splnil	Existují nějaké důkazy o splnění a přístupu k definované procesní vlastnosti. Nicméně některé aspekty hodnoceného mohou být nepředvídatelné.	>15% až $\leq$ 50%
L	Převážně splnil	Existují důkazy o systematickém přístupu a značné splnění hodnocené procesní vlastnosti, avšak existují slabiny, které jsou s tímto procesem spjaty.	>50% až $\leq$ 85%
F	Plně splnil	Je dostatek důkazů o splnění dané procesní vlastnosti, mohou existovat nedostatky, které však nejsou závažné.	>85% až $\leq$ 100%

Tabulka 2: Stupnice splněnosti jednolivých procesů dle ISO/IEC 33020 [1]

Tato stupnice má ještě následné rozdělení, které přesněji určuje na kolik procent byl daný proces splněn. Upřesňující rozdělení je vytvořeno pro hodnocení *částečně splnil* (P) a *převážně splnil* (L) viz Tabulka 3.

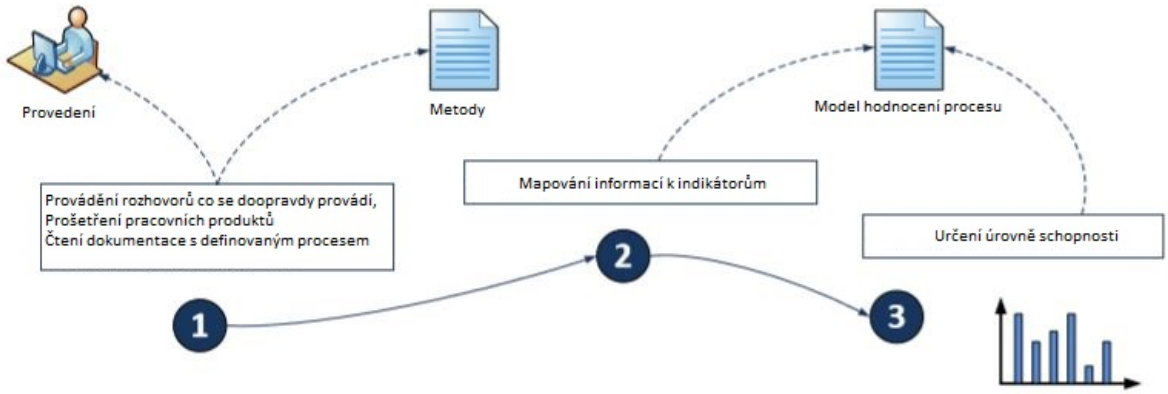
P -	Částečně splnil -	>15% až $\leq$ 32,5%
P +	Částečně splnil +	>32.5% až $\leq$ 50%
L -	Převážně splnil -	>50% až $\leq$ 67,5%
L +	Převážně splnil +	>67,5% až $\leq$ 85%

Tabulka 3: Upřesňující tabulka hodnocení dle ISO/IEC 33020 [1]

Upřesňující hodnocení zlepšuje organizacím pohled na to, jak moc by se měly na daný proces zaměřit při jeho vylepšení, aby při následném hodnocení mohl dosáhnout na hodnocení plně splnil (F).

2.4 Příklad hodnocení review procesu

Jak jsem již zmínil, celý proces hodnocení všech procesů probíhá „výslechem“ jednotlivých lidí v organizaci, kteří vlastními slovy popisují, jakým způsobem se vykonává daný proces. Hodnotitel tedy musí správně identifikovat z jejich „výpovědi“ jestli vykonávání tohoto procesu se drží standartu či nikoliv. Proces je znázorněn na Obrázku 3.



Obrázek 3: Proces hodnocení úrovně procesu schopnosti [1]

Jako příklad procesu jsem zvolil proces Joint review. Tento proces se řadí mezi podpůrné procesy a ve standartu jsou označeny zkratkou SUP. Každý proces obsahuje několik bodů, které musí být splněny. Těmito body se hodnotitel řídí a uděluje hodnocení na základě Tabulky 2. a 3. Jednotlivé body, které proces musí dodržovat viz Tabulka 4.

Označení	Popis
SUP.4.BP1	Definovat hodnocené prvky: Dle potřeb projektu identifikovat plán, který pojme vhodnou část projektu, umístění a pracovníky, kteří se budou podílet ne revidování a stanoví se kritéria hodnocení.
SUP.4.BP2	Ustálení mechanismu pro zpracování revize: Nastolit mechanismy, které umožní přístup k výsledkům revize všem zúčastněným. Dále musí být zaznamenány problémy zjištěné při kontrolách.
SUP.4.BP3	Příprava revize: Shromáždit, naplánovat, připravit a předat veškeré materiály týkající se revize.
SUP.4.BP4	Společné provádění revize: Společně řídit technické přezkoumání podle naplánování. Následně zaznamenat výsledky přezkoumání.
SUP.4.BP5	Distribuovat výsledky: Distribuovat vzniklé záznamy mezi všechny účastníky vývoje
SUP.4.BP6	Určit následný postupy pro výsledky revize: Analyzovat výsledky a následně navrhnout opatření a vhodně jednotlivá opatření upřednostňovat.
SUP.4.BP7	Sledovat následná opatření: Zajistit sledování vzniklých opatření pro úpravu nesrovnalostí až po uzavření.
SUP.4.BP8	Identifikovat a zaznamenat problém: Identifikovat a zaznamenat všechny problémy, které byly zjištěny během revize, dle zavedených mechanismů.

Tabulka 4: Body, které musí být splněny při hodnocení procesu joint review [1]

Přesné definice jednotlivých bodů jsou k nahlédnutí ve standartu [1]. Hodnotitel na základě výpovědí lidí účastnících se hodnotících „výslechů“ ohodnotí dle stupnice jednotlivé body, které

dle standartu musí být splněný. Výsledné hodnocení procesu je pak tvořeno aritmetickým průměrem jednotlivých hodnocení bodů nebo jiným vzorcem. Výsledkem správného nasazení tohoto procesu vzniká:

1. Řízení a technické revize vznikají dle potřeb projektu.
2. Stav a produkty jsou hodnoceny podle bodů v Tabulce 4. společně se všemi účastníky.
3. Výsledky jsou dodány všem účastníkům vývoje
4. Jednotlivé body z revize jsou sledovány až k jejich ukončení
5. Problémy jsou identifikovány a následně zaznamenány

Můj vedoucí mi poskytl konkrétní záznam hodnocení procesu, který slouží k výukovým účelům a je prováděn na školení hodnotitelů Automotive SPICE. Ukázkový záznam viz Obrázek 4.

SWE.6	Software Qualification Test					Purpose Outcomes	
Attributes	Attribute Rating	Practice	Practice Rating 0..100%	NPLF % Value	NPLF	Evidence	Weakness (Description of gap in implementation)
PA 2.1	P- (31%)	GP 2.1.1	23	23	P-	standard process define purpose of the process. The current status of software meets the defined requirements is shown by the colour in the test matrix in the test manual	not mesureable objectives are set
		GP 2.1.2	23	23	P-	basic plan is set up	
		GP 2.1.3	23	23	P-	monitored at the way that after due date schedule is not met	
		GP 2.1.4	23	23	P-	adjusted plan a added more resources	
		GP 2.1.5	41	41	P+	roles assigned to the project team, but not clearly defined	not clearly defined authorities (RACI table)
		GP 2.1.6	23	23	P-	at the beginnig not enough resourrces - testers not properlu iintegrated to the team. Added lter on at later project phase	
		GP 2.1.7	58	58	L-	stored in at the same place and everybody know ewhere to find it. Since there is not too much project members, the cuminication is assured, but not systematicaly.	no regular meetings, or regular briefing
PA 2.2	L+ (75%)	GP 2.2.1	100	100	F	templates, guidelines and checklists for our documents.	
		GP 2.2.2	100	100	F	The documents are stored and version controlled in the Stages portal (Teamwork portal) and MKS SI configuration management system.	
		GP 2.2.3	100	100	F	foloowed and used	
		GP 2.2.4	0	0	N	no evidence	

Obrázek 4: Proces hodnocení úrovně procesu schopnosti

Toto hodnocení bylo provedeno pro proces ze skupiny procesů softwarového vývoje (SWE). Konkrétně pro proces testování kvalifikace softwaru.

Tento proces zajišťuje, aby integrovaný software byl testován a byly vytvořené záznamy o tom, že celý software splňuje požadavky, které byly definovány.[1]

Záznam, který je na Obrázku 4. je tvoře pro druhou úroveň procesu schopnosti a je patrné, že převážná většina bodů je splněna jen z část (hodnocení P+ a P-) viz Tabulka 3. Lze tedy tvrdit, že v organizaci, která by měla tento proces takto definován, by nedosáhla na druhou úroveň daného procesu. Konkrétní požadavky, které musí být splněny pro druhou úroveň, jsou k nalezeni viz [1].

3 Software reviews

V této kapitole se budu věnovat softwarovým revizím, rozeberu jaké druhy revizí software se provádí a jaký je jejich účel.

3.1 Důvod revizí

S narůstajícím množstvím zákaznických požadavků a stále se zvětšující komplexností všech softwarových produktů vzniká problém zpracovat všechny požadavky tak, aby nebyly žádné opomenuty a mohl se vždy dodat software, který zákazníka maximálně uspokojí. S tímto problémem právě pomáhají softwarové revize, které zlepšují přehlednost o tom, které požadavky již byly splněny, popřípadě se odhalila chyba, která při vývoji mohla vzniknout. To dopomáhá k tomu, aby se mohlo průběžně zjišťovat, v jaké kvalitě jsou jednotlivé požadavky zpracovány a zdali jsou splněny kvalitativní standarty.

Hlavním smyslem softwarových revizí je tedy zlepšování softwarové kvality. Tyto revize jsou založeny na dokumentech, které byly vytvořeny během vývoje daného software, jakými jsou uživatelské požadavky, analýza a návrh, zdrojový kód, plány testů, uživatelská dokumentace a další. Všechny tyto dokumenty by měly být během vývoje revidovány, aby se kvalita softwaru co nejvíce zvýšila.[2] Využívání softwarových revizí při vývoji přináší výhody.

- Zlepšuje produktivitu vývojového týmu
- Snižuje množství vzniklých chyb ve finální verzi produktu, která se předává zákazníkovi
- Méně chyb ve finální verzi produktu snižuje náklady na následnou opravu těchto chyb
- Snižuje počet chyb při vývoji software
- Zvyšují pravděpodobnost včasného dodání softwaru zákazníkovi

3.2 Typy revizí

Jelikož revidování všech artefaktů při tvorbě software je rozsáhlá disciplína, existuje také více typu revizí, které jsou vykonávaný různými členy vývojového týmu. Tyto revize se rozdělují na tři typy.[3]

1. **Software Peer review**
   Software peer review je proces sloužící ke zhodnocení technické stránky vytvářeného produktu (software) a posouzení jeho kvality člověkem, který na tomto produktu (software) pracuje. Toto posouzení provádí společně s dalšími členy vývojového týmu. Tímto procesem se předchází k zavedení chyb do produkční verze produktu (software). [3] Existuje více software peer review, které se soustředí na různé artefakty vyvíjeného produktu.

- **Walkthrough:** Je jeden ze způsobu k zajišťování kvality vyvíjeného software. Při provádění této revize se dbá na zjištění, zdali je vyvíjený produkt vyvíjen dle požadavku zákazníka a zároveň jestli tento produkt poskytuje očekávané výsledky. Hlavní náplní walkthrough je odhalit chyby, které se ve vyvíjeném produktu nachází, avšak jejich následná oprava je ponechána na autorovi chyby, popřípadě jiném členu vývojového týmu. V ideálním případě by se mělo walkthrough provádět ve všech fázích vývojového cyklu. Účastníci toho revidováni softwaru se mohou měnit. [4]

  Celý proces walkthrough je veden osobou (programátor nebo designer), která prochází společně s vývojovým týmem i ostatními účastníky vývoje vyvíjený software. Kompletní proces je rozdělen do podprocesů, které tvoří celé walkthrough.

  1. **Příprava managementu -** zajišťují, aby celý proces proběhl dle standardních postupů. Plánuje se čas a zdroje potřebné k provedení.[7]
  2. **Plánování -** vytvoří se tým, který bude provádět revizi. Naplánuje se schůzka a její místo vykonání. Připraví se materiály pro přípravu všech účastníků.[7]
  3. **Úvodní schůzka -** shrnutí všech bodů, které budou přezkoumány během přezkoumání.[7]
  4. **Příprava -** lídr rozešle připravené dokumenty mezi všechny účastníky revize. Ti následně provádějí přezkoumání samostatně a všechny poznatky zaznamenávají do dokumentu, který následně pošlou lídrovi revize. Veškeré zaznamenané poznatky jsou poté diskutovány během přezkoumání.[7]
  5. **Přezkoumání -** účastníci pokládají otázky ohledně nedostatku, které nalezli během přípravné fáze a navrhují řešení a úpravy. Veškeré tyto poznatky, které během schůzky vznikly jsou zaznamenány a výstupem je report celé schůzky.[7]
  6. **Přepracování/následné schůzky -** verifikují se všechny poznatky a předají se k přepracování.[7]

  Všichni účastníci walkthrough revize mají svou roli. U projektů menšího rozsahu může jeden účastník také zastávat více rolí najednou, těchto rolí je celkem pět. [4]

  1. **Autor** - Osoba, která žádá o revizi svého produktu nebo jeho části, která byla vytvořena a odladěna tak, aby neobsahovala nahodilé vady nebo chyby. Autor při průchodu již pouze reaguje na případné dotazy účastníků revize.[4]
  2. **Přednášející** - Osoba, která obvykle program pro danou revizi a prezentuje produkt, který je přezkoumáván. Měl by být s produktem obeznámen v ideálním případě by tuto roli měl zastávat někdo z týmu, který na projektu pracuje.[4]
  3. **Moderátor** - Osoba, která se stará o to, aby byl dodržen program, který vytvořil přednášející. Snaží se, aby se všichni hodnotitelé zapojili do revize. Osoba v této roli může zároveň zastávat i roli zapisovatele.[4]
  4. **Hodnotitelé** - Osoby hodnotící produkt, který byl předložen k revizi. Tyto osoby hodnotí jestli se dodržují standardy vývoje a zdali jsou splněny požadavky spjaté s tímto projektem.[4]

5. **Zapisovatel** - Osoba, která zaznamenává průběh revize, zapisuje poznámky o chybách, které byly objeveny během revize. Dále zaznamenává nápady a nevyřešené dotazy, které vznikly během revidování produktu. Tuto roli nesmí zastávat nikdo z hodnotitelů.[4]

Výsledkem tohoto procesu je zdokumentování všech poznatků, které byly objeveny. Důraz při tvorbě této dokumentace se klade především na:

- Jestli zkoumaný produkt splňuje cíle vývoje
- Nesmí existovat žádné nejasnosti nebo opomenutí některého požadavku
- Dokument je jednoduše pochopitelný

Toto revidování software se používá hlavně při vývoji, kde je dbáno na co nejvyšší kvalitu. Pomáhá včasně odhalovat chyby v software, tím je zajištěno snižování množství chyb, které mohou být objeveny v produkční verzi.[4] Z toho vyplývají hlavní výhody jakými jsou:

- Nižší finanční a časové náklady na případnou opravu odhalených chyb
- Vývojáři jsou průběžně obeznámení s kvalitou jimi vyvíjeného software
- Účastní tohoto revidování systému získávají znalosti a zkušenosti od ostatních účastníků

- **Code review:** Je dalším způsobem revize. V tomto případě je však veškerá pozornost soustředěna na programový kód. Autor tohoto kódu (programátor) společně s kolegy z vývojového týmu (programátoři a testeři) prochází konkrétní části kódu a hodnotí, jestli je kód psán dle zavedených postů a standardů. Tento průchod kódem pomáhá k odhalení technických nedostatků, které při vývoji mohly být zaneseny do kódu.
  - Problémy s uvolněním nepotřebné paměti (memory leaks)
  - Přetečení paměti
  - Bezpečnostní problémy

  Dále je posuzovaná kvalita komentářů a dokumentace jednotlivých metod (jestli je správně popsáno jaké jsou vstupy a jaký je výstup).[5]

  Tento proces může být vykonáván manuálně (studování kódu) nebo automatizovaně za pomocí specializovaných programů. Existuje několik metod, jak provést tento proces revidování kódu.

  - **E-mail pass around**
    Je flexibilní přístup k provedení této revize. Revidovaný kód je zaslán e-mailem účastníkům revize, kteří provedou hodnocení daného kódu a zpětně reagují e-mailem zpět autorovi, který kód zaslal. Zde však může nastat problém, že e-mail se může ztratit mezi množstvím dalších e-mailů a tak může docházet k časovým prodlévám při hodnocení.[5]

– **Over the shoulder**
Při revidování tímto způsobem autor kódu přizve jednoho svého spolupracovníka z vývojového týmu a společně s ním prochází tento kód. Při průchodu napsaného kódu svému kolegovi dodává detailnější informace ohledně kódu.[5] Takto téměř eliminuje proces studování kódu svému kolegovi a tím je ušetřen čas a zároveň dochází k lepšímu porozumění kódu.

– **Tool Assisted**
Metoda využívající specializované programy, které jsou možné integrovat do různých vývojových prostředí. Tyto programy slouží k poskytnutí statistický přehled o kódu a poskytují jeho metriky.[5]

- **Inspection:** Hlavním cílem inspekcí softwaru je odhalovat chyby (bugy), které mohou vznikat v softwaru, který je vyvíjen. Na tomto procesu se podílí členové týmu, kteří společným úsilím se snaží maximalizovat počet odhalených chyb a tím zlepšit kvalitu vyvíjeného softwaru. Inspekce doplňuje proces testování, protože vyvíjený software nemusí být spuštěn. To umožňuje ověřitelnost i neúplných verzi softwaru a během této činnosti můžou být validovaný reprezentace, jako je například UML.

  Inspekce zahrnuje do svého průběhu členy týmu z různých prostředí. Tito členové týmu následně provádějí revizi zdrojového kódu. Hledají problémy a defekty, které mohli vzniknout při vývoji a následně tyto problémy a defekty, které odhalili popisují na schůzce, která je součástí celého procesu. Při této kontrole se využívají kontrolní seznamy, které obsahují záznamy o běžných chybách, které vznikají při vývoji a jsou běžné v dané doméně vývoje. Tyto seznamy mohou být založeny na příkladech z knih nebo vycházet z praktických znalostí. Kontrolní seznamy by měly být různé, pro různé programovací jazyky. Hlavním důvodem toho je, že každý programovací jazyk má své specifické chyby, které nejčastěji nastávají při vývoji. V Tabulka 5. obsahuje příklad, jak by mohl takový kontrolní seznam vypadat.[2]

Druh chyby	Kontrola
Datová	- Jsou všechny proměnné inicializovány před použitím? - Je možnost přetečení paměti?
Řídící	- Je každý cyklus správně ukončen? - Jsou správně vytvořeny podmínky (if statement)?
Vstup/výstup	- Jsou všechny vstupní proměnné využity? - Může být problém neočekávaný vstup?
Rozhraní	- Mají funkce správný počet parametrů při volání? - Jsou parametry ve správném pořadí ?

Tabulka 5: Příklad kontrolního seznamu [2]

Celý proces inspekce má jednotlivé fáze, které je potřeba provést, aby celý proces mohl kompletně proběhnout.

1. **Plánování -** naplánují se činnosti, které se provedou během inspekce.
2. **Přehledová schůzka -** na této schůzce se předávají informace ohledně produktu, který se bude přezkoumávat.
3. **Příprava -** identifikování možných problémů a defektu v produktu.
4. **Hlavní schůzka -** procházejí se veškeré body, které jsou předmětem kontroly a inspektoři se snaží poukázat na chyby, které nalezli během přípravné fáze.
5. **Přepracování -** provádí se na základě hlavní schůzky, provádí se opravy chyb, které byly nalezeny a zaznamenány během hlavní schůzky.
6. **Následné přezkoumání -** tyto změny jsou přezkoumány autorem.

Hlavní výhodou provádění softwarové inspekce je odhalování chyb v průběhu tvorby softwaru, tím se následně předchází dalším finanční výdajům na opravu dané chyby, například již z produkční verze softwaru.

- **Technical review:** Důvodem technických revizí software je zhodnocení vyvíjeného softwarového produktu, týmem kvalifikovaných pracovníků, kteří přezkoumávají vhodnost použití technických prostředků k vývoji. Zároveň se snaží zajistit, aby nevznikly nesrovnalosti se specifikacemi a normami. Tímto se dopomáhá k zajištění, aby vyvíjený software odpovídal definovaným specifikacím, dodržování postupů pro vývoj daného projektu a zároveň, aby změny, které nastávají během vývoje byly správně implementovány.[7] Artefakty, které spadají pod tento proces jsou

  a) Specifikace požadavků
  b) Uživatelská dokumentace
  c) Dokumentace testovacích případů
  d) Manuál údržby

  Pod technické revize mohou spadat i další dokumenty, které nejsou výše zmíněné viz IEEE Std 1028-1997.

  Celý proces technických revizí má definované role, které by se měly podílet na úspěšném provedení celého přezkoumání.

  1. **Tvůrce rozhodnutí -** je osoba, kvůli které se celá revize provádí. Rozhoduje o tom, jestli bylo dosaženo cílů během přezkoumání.[7]
  2. **Review lídr -** je odpovědný za celý proces revize. Zajišťuje administrativní úkoly, které předcházejí celému procesu a následně zajišťuje, aby celá revize byla provedena řádným způsobem.[7]
  3. **Zapisovatel -** zaznamenává všechny nalezené nesrovnalosti a rozhodnutí, které vzniknou celým revizním týmem.[7]
  4. **Technický personál -** aktivně se podílí na hodnocení softwarového produktu.[7]

  Celý proces je rozčleněn do podprocesů, které slouží k vykonání potřebných činností, aby celá technická revize proběhla dle požadovaných náležitostí.

1. **Plánování -** během této činnosti se přidělí jednotlivým členům revizního týmu, za jaké úkoly budou odpovídat, určí se, kde bude revize probíhat. Rozešlou se veškeré materiály všem členům, které se budou přezkoumávat.[7]
2. **Přehled revize -** provádí se na vyžádání review lídra. Na této schůzce je prezentován, jak bude celý proces probíhat. Tento přehled může být také součástí až při přezkoumávání.[7]
3. **Představení softwarového produktu -** na této schůzce, je představeno jaké dokumenty nebo jiné náležitosti budou součástí revizního procesu.[7]
4. **Příprava -** všichni účastníci týmu, který provádí revizi prozkoumají jednotlivé dokumenty, které podléhají revizi. Pokud naleznou nějaké nedostatky, zaznamenají je a pošlou je review lídrovi.[7]
5. **Přezkoumání -** může být provedeno v jedné nebo více schůzkách, záleží na rozsahu revidovaných materiálů. Hodnotí se jestli nejsou v softwaru chyby, určí se jestli je produkt kompletní, jestli jsou dodrženy standarty vývoje, změny v softwaru jsou korektně implementovány. Toto vše je zdokumentováno.[7]
6. **Přepracování/následné přezkoumání -** přepracují se odhalené nedostatky a následně review lídr verifikuje jejich korektní přepracování.[7]

2. **Software Management review**

   Hlavním cílem software management review je sledování postupu vývoje, určení v jakém stavu se nachází projekt dle plánu. Dále slouží k průběžnému potvrzování splnění požadavků, dle těchto kriterií jsou následně vhodně přiděleny zdroje pro postupný vývoj softwarového produktu. Následně díky tomuto procesu může být změněn rozsah projektu tak, aby zákazník byl co nejvíce spokojen. Management review je prováděno pracovníky na manažerských postech, kteří jsou přímo odpovědní za daný projekt. [7] Produkty, které podléhají takovéto revizi jsou například

   a) Zpráva o postupu práce (Progress report)
   b) Plány řízení rizik (Risk management plan)
   c) Analýza softwarového produktu
   d) Plán instalaci
   e) Plán údržby

   Těchto produktů, které mohou být zahrnuty v management review, je více. Je možné je nalézt ve standartu IEEE Std 1028-1997. Celý proces management review zahrnuje role, které se snaží obstarat jeho vykonání.

   1. **Tvůrce rozhodnutí -** je osoba, pro kterou je celá revize vykonávána. Má hlavní slovo, jestli byly splněny cíle revize, či nikoliv.[7]

2. **Review lídr -** osoba, která by měla být zodpovědná za administrativní úkoly, které předchází celé revizi. Dále by měla odpovídat za plánování a přípravu celé revize. Dohlíží na to, aby revize probíhala správně a splňovala svůj cíl.[7]
3. **Zapisovatel -** dokumentuje veškeré nesrovnalosti, které jsou během revize nalezeny.[7]
4. **Personál z managementu -** zajišťuje, aby revize byla prováděna dle požadavků a zároveň byly dodrženy standarty.[7]
5. **Technický personál -** dodává veškeré nezbytné informace manažerům, aby mohli plnit své pracovní povinnosti během revize. [7]

Celý proces management review je tvořen podprocesy, které je nutné vykonat před samotným začátkem přezkoumání, tak následně po přezkoumání. Těchto podprocesů je celkem šest a jsou vykonány jednotlivými rolemi, které jsou zmíněny výše.

1. **Příprava managementu -** naplánuje a poskytne finanční prostředky, které jsou potřebné pro vykonání revize. Zajistí, aby tým, který bude provádět revizi byl proškolen a jejich znalosti ohledně softwarového produktu byly co největší.[7]
2. **Plánování revize -** naplánuje se čas a vytýčí se zdroje potřebné pro provedení revize. Přiřadí se úkoly jednotlivým členům revizního týmu, za které budou odpovídat. Jednotlivé materiály, které jsou potřebné k dané revizi se doručí všem účastníků, kteří se budou podílet na přezkoumání.[7]
3. **Přehled revize -** může být součástí přezkoumání, ale může být vedeno jako samostatná schůzka, kde se provede přehled, jaké aktivity se budou přezkoumávat.[7]
4. **Příprava -** členové týmu, kteří se podílejí na přezkoumání musejí projít softwarový produkt a další dokumenty, které jsou revidovány. Nalezené chyb pak následně zaznamenají a pošlou lídrovi revize.[7]
5. **Přezkoumání -** revidují se cíle, které byly vytyčeny během plánování, hodnotí se zdali projekt probíhá podle plánů. Revidují se chyby, které byly objeveny během příprav.[7]
6. **Přepracování/následné přezkoumání -** verifikuje se zdali jsou jednotlivé chyby, které byly na schůzce objeveny přepracovány.[7]

3. **Software Audit review**

Důvodem vykonání softwarového auditu je vytvoření nezávislého posudku softwarového produktu, který je vyvíjen. Tato revize se zaměřuje především na hodnocení, jestli je produkt vyvíjen v souladu s platnými předpisy a normami, jsou-li splněny pokyny a plány, které jsou definovány. Předmětem softwarového auditu může být cokoliv co má něco společného s vyvíjeným produktem, který podléhá danému audit.[7] Tyto předměty jsou například

a) Smlouvy

b) Zprávy z výše zmíněných druhů revizí

c) Zdrojové kódy

d) Řízení rizik

e) Uživatelské manuály

Více druhů předmětů, které podléhají softwarovému auditu lze nalézt ve standartu IEEE Std 1028-1997. Na počátku celého auditu by měla být provedená schůze, kde auditoři a organizace, pro kterou bude daný audit vykonán, dohodnou podmínky a jednotlivé předměty, které budou v auditu zahrnuty. V procesu se objevují role, které vykonávají specifické činnosti v průběhu auditu.[7] Rolí v softwarovém auditu je celkem pět.

1. **Vedoucí auditor -** je odpovědný za celý audit, připravuje plán auditu, řídí tým auditorů, připravuje report auditu.[7]
2. **Zapisovatel -** dokumentuje veškeré poznatky, které jsou během auditu učiněny a doporučení ohledně těchto poznatků.[7]
3. **Auditoři -** přezkoumávají veškeré předměty, které podléhají auditu. Zaznamenají veškeré poznatky, které učinili během přezkumu. Musejí být nezávislí, aby mohli provádět co nejobjektivnější posouzení.[7]
4. **Iniciátor -** podává návrh k provedení auditu, definuje rozsah auditu, kritéria a kdo audit provede.[7]
5. **Organizace podléhající auditu -** poskytne veškeré materiály, které jsou potřebné k vykonání auditu.[7]

Aby audit byl vykonán co nejpreciznějí, je rozdělen do jednotlivých činností, které tvoří celý jeho proces.

1. **Příprava managementu -** manažeři zařizují, aby audit byl proveden dle standardních postupů. Určí čas vykonání auditu a jeho rozsah.[7]
2. **Plánování auditu -** Blíže se specifikuje rozsah auditu a jeho účel, vhodně se přiřadí zdroje pro vykonání. Rozdělí se jednotlivé odpovědnosti mezi tým, který bude provádět audit.[7]
3. **Úvodní schůzka -** je mezi týmem, který bude audit provádět a organizací, která o audit požádala. Slouží jako přehledová schůzka, která informuje auditorský tým o rozsahu auditu, produktech na kterých budou provádět přezkoumání. Zpřístupní se veškeré informace nezbytné pro audit.[7]
4. **Příprava -** Organizace je informována o provedení auditu. Informování probíhá písemnou formou, kde je popsáno, co vše bude zahrnovat audit, který bude proveden. Hlavní důvod tohoto oznámení je, aby veškerý personál a materiály podléhající auditu byl v daný den dostupné.[7]

5. **Přezkoumání -** během přezkoumání si sbírají důkazy a poznatky, které jsou následně analyzovány a dokumentovány.[7]
6. **Následné schůzky -** definuje se jaké akce pro opravu budou provedeny a následně se iniciují.[7]

Výsledkem celého auditu je zpráva (report), které obsahuje sumarizaci provedeného auditu (důvod a rozsah auditu, identifikaci softwaru, který podléhal auditu, hodnotící kriteria, shrnutí celého auditu a nálezů, které nebyly v souladu s provedeným auditem a další).[7]

4 Průzkum dostupných nástrojů pro podporu reviews

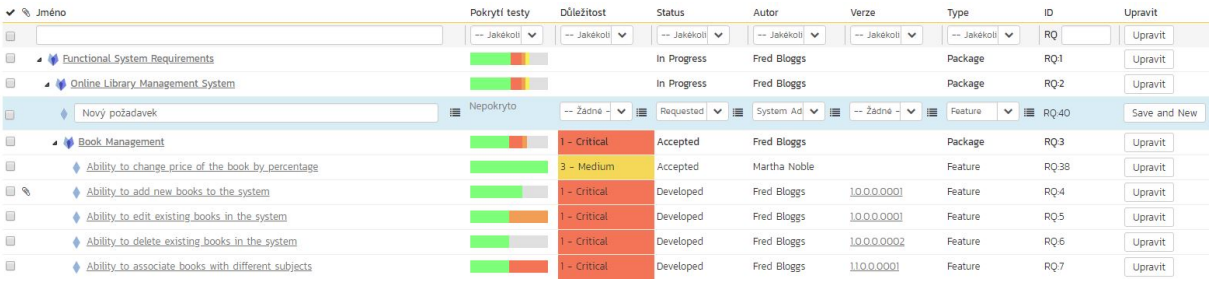
V této kapitole se budu zabývat hledáním již hotových řešení, které jsou v produkční verzi na trhu. Hlavními aspekty, které budu v těchto programech hledat je správa požadavků (requirement management) a následné možností revidování těchto požadavků a dalších artefaktů, které jsou nedílnou součásti vývoje. Softwary jsou vybrány na základě této stránky [8] a následném přečtení popisu softwaru na webové stránce, kde je prezentován.

4.1 Spira

Systém vyvinutý společností Inflectra Corporation. Celý systém je rozdělen do několika subsystému, které jsou na sobě nezávislé a dokáží se integrovat s jinými systémy, nebo vývojovými prostředími. V mém výzkumu jsem zahrnul celkem tři subsystémy, které tato firma nabízí a mohli potenciálně obsahovat funkcionalitu, kterou vyhledávám.

Spira test, Spira team, Spira plan

Spira test, systém který je určen především pro testery a vývojáře. Jeho ovládání je velmi jednoduché a intuitivní. Nabízí velmi přehlednou správu požadavků, která je velmi jednoduše spravovatelná. Vkládání požadavků na vývoj vypadá viz Obrázek 5.



Obrázek 5: Přidání požadavků na vývoj do systému Spira test

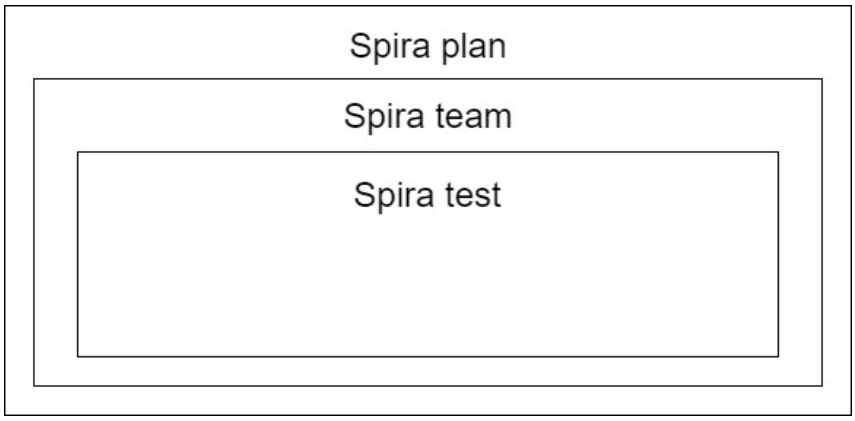
Na obrázku lze vidět, že systém sleduje jakou má daný požadavek důležitost, v jaké se nachází fázi, kdo je za něj zodpovědný, v jaké verzi byl přidán a o jaký typ požadavku se jedná.

- **Důležitost -** Kritická, vysoká, střední, nízká
- **Status -** Vyžádaný, podléhá revizi, odmítnutý, přijatý, plánovaný, ve vývoji, vyvinutý, testovaný, kompletní, zastaralý
- **Typ -** Prvek návrhu, vlastnost, kvalita, use case

Celý systém ve velmi jednoduché a přehledné formě dokáže sledovat postup vývoje softwaru. Nicméně vlastní možnost revidování jednotlivých požadavků či jiných artefaktů vývoje neumožňuje. Možnost, jak by se dala táto správa provádět, je tvořit se revizní dokumenty sám, v externí aplikaci například v excel nebo word a následně tyto soubory nahrávat do systému. To však ale

nezaručí vysledovatelnost, jednotlivých požadavků a artefaktů ke konkrétní revizi, ve které byly zahrnuty.

Spira team a spira plan sice rozšiřují funkčnost především co se týče plánování a reportováni informací, které se týkají projektu, ale žádné funkce spojené s prováděním revizí jako je vytvoření revizního formuláře a následné přiřazení požadavků nebo artefaktů, které mají být revidovány neobsahují. Funkčnost těchto systému v závislosti na sestavení lze vidět viz Obrázek 6.



Obrázek 6: Funkčnost jednotlivých systému Spira

Celý systém bych hodnotil jako velmi přehledný a jednoduchý, avšak pro mou diplomovou práci nesplňuje hlavní kritéria a to je podpora reviews.

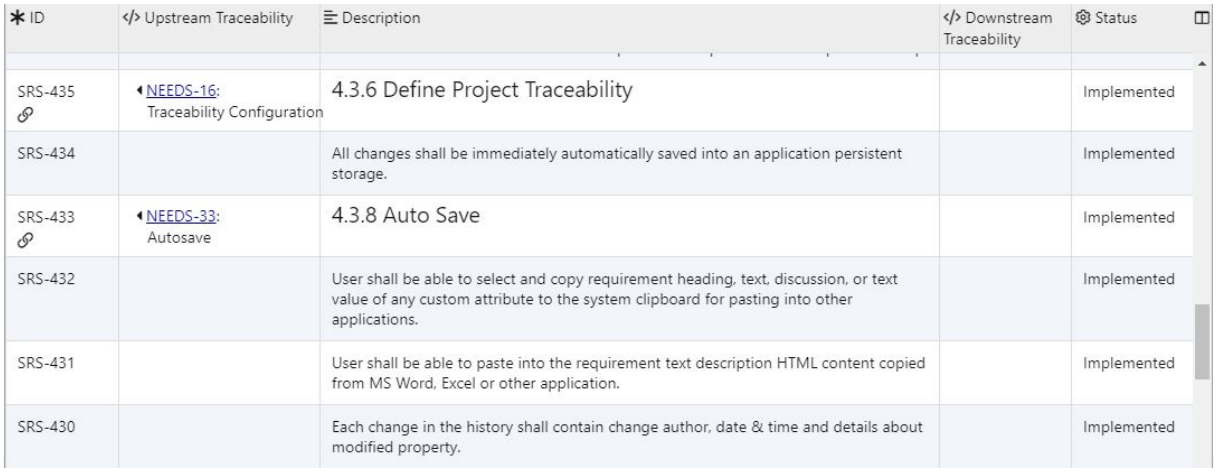
4.2 ReqView

Software je vyvinut a spravován společnosti Eccam. Nástroj disponuje jak webovým klientem, tak desktopovým rozhraním a tyto dvě rozhraní jsou svou funkcionalitou téměř totožná, drobnosti ve kterých se liší jsou dohledatelné [6]. Využil jsem možnosti 14 dní zkušební verze Pro, jejíž cena za roční aktivaci činí 300$ za jednu uživatelskou licenci.

Reqview nabízí startovní projekt, který je tvořen požadavky na software Reqview. Celé prostředí je členěno do jednotlivých dokumentů, ze kterých se celý projekt skládá. Vzhled těchto dokumentů je koncipován jako tabulka a její vzhled je flexibilně definovatelný. Jednotlivé sloupce, pro které si uživatel může vytvořit vzhled sám je jejich tvorba obtížná a probíhá pomocí zapisování atributů v json formátu viz Obrázek 7.

Obrázek 7: Definování sloupce v ReqView

Tato definice konkrétně vytvoří jeden sloupec s názvem Status a bude obsahovat tři možnosti, mezi kterýma lze vybírat při jeho vyplňování. Takto lze vytvořit celý dokument, do kterého následně lze vyplnit artefakty, které se týkají vývoje. Každý dokument musí mít své unikátní ID. Toto ID vytvoří uživatel sám a následně při zadávání jednotlivých položek do dokumentu se tyto položky číslují a jejich prefix tvoří ID dokumentu. Ukázka dokumentu se softwarovou specifikací požadavků viz Obrázek 8.



ID	Upstream Traceability	Description	Downstream Traceability	Status
SRS-435	NEEDS-16: Traceability Configuration	4.3.6 Define Project Traceability		Implemented
SRS-434		All changes shall be immediately automatically saved into an application persistent storage.		Implemented
SRS-433	NEEDS-33: Autosave	4.3.8 Auto Save		Implemented
SRS-432		User shall be able to select and copy requirement heading, text, discussion, or text value of any custom attribute to the system clipboard for pasting into other applications.		Implemented
SRS-431		User shall be able to paste into the requirement text description HTML content copied from MS Word, Excel or other application.		Implemented
SRS-430		Each change in the history shall contain change author, date & time and details about modified property.		Implemented

Obrázek 8: Příklad dokumentu se specifikací požadavků

U každého dokumentu je diskuzní sloupec, do kterého mohou přispívat uživatelé. Toto by mohlo částečně splňovat funkcionalitu revidování. Následně je možné vytvořit vlastní dokument pro určitou revizi a do něj zařadit jednotlivé artefakty, které budou v revizi zahrnuty. Toho však lze dosáhnout pouhým kopírováním jednotlivých artefaktů z jiných dokumentů. Automatizovaný proces jsem v tomto systému nenalezl. Dalším problémem bylo přiřazení rolí pro jednotlivé přispěvatele do revize, tuto funkčnost jsem v systému také nenalezl. Celý systém splňuje funkcionalitu, kterou hledám jen z části.

4.3 Enterprise architect

Nástroj, který je vyvinut a spravován společností Sparx Systems. Tento nástroj je velmi rozsáhlý a je vytvořen pro správu každé domény vývoje softwaru (Byznys, softwarový návrh, systémový návrh, architektura). Celý systém je rozdělen do čtyř edicí, kdy každá edice má rozsáhlejší funkcionalitu. Přehled edicí viz tabulka 6. Enterprise architect nabízí 30 denní zkušební verzi

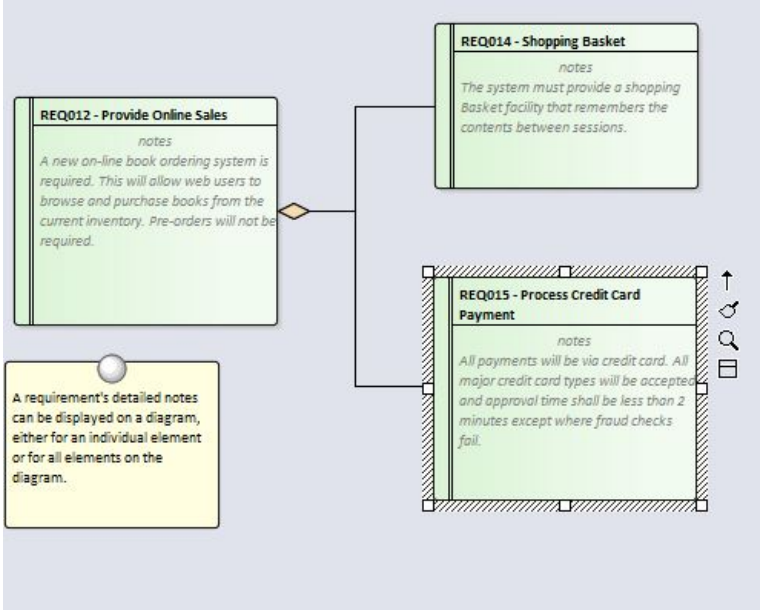
Edice	Cena za licenci
Proffesional	229$
Corporate	299$
Unified	499$
Ultimate	699$

Tabulka 6: Ceník licencí Enterprise architect

všech svých produktů, mezi kterými lze libovolně přepínat po dobu zkušební verze. Pro můj výzkum jsem si zvolil nejrozsáhlejší verzi Ultimate.

Enterprise architect nabízí startovní projekt, který je poměrně rozsáhlý co se týče různých uživatelských požadavků a další artefaktů. Zaměřil jsem se především na specifikací požadavků a možnost revidování artefaktů.

Zadávání požadavků je prováděno pomocí diagramů, které jsou v programu již předdefinovány. Jednou z možností je tvořit diagram závislostí jednotlivých požadavků na sobě viz Obrázek 9.



Obrázek 9: Příklad zadávání požadavků zákazníka

Další možnost jak specifikovat požadavky je pomocí náhledu na jednotlivé požadavky zákazníka viz Obrázek 10.

Item	Stereotype	Status	Difficulty	Priority
1 Add New Titles This defines the process for adding new titles.		Proposed		
2 REQ019 - Manage Inventory The system MUST include a complete inventory management facility to store and track stock of books for the on-line bookstore.	Functional	Approved	Medium	Medium
2.1 REQ122 - Inventory Reports Inventory reports are required that detail the available stock for each item including back orders. Future stock level reports should be able to predict the quantity of stock at a specified future date.	FunctionalRequirement	Proposed	Medium	Medium
2.2 REQ023 - Store and Manage Books A book storage and management facility will be required.	Functional	Validated	Medium	Low

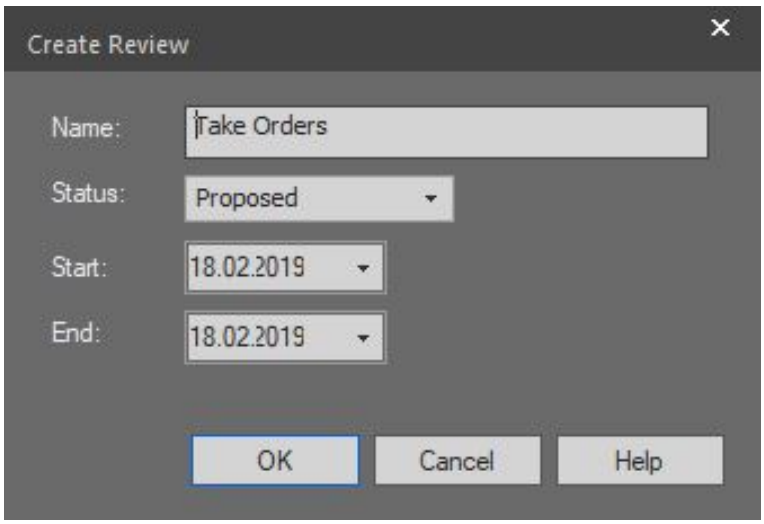
Obrázek 10: Příklad zadávání požadavků zákazníka pomocí náhledu

U zákaznických požadavků je opět možnost nastavit atributy, které blíže specifikují daný požadavek vývojářům.

- **Prioritu** - vysoká, střední, nízká
- **Obtížnost** - vysoká, střední nízká
- **Status** - schválený, implementovaný, validovaný, vyžádaný
- **Typ** - funkční, výkonnostní, vzhledový, a další

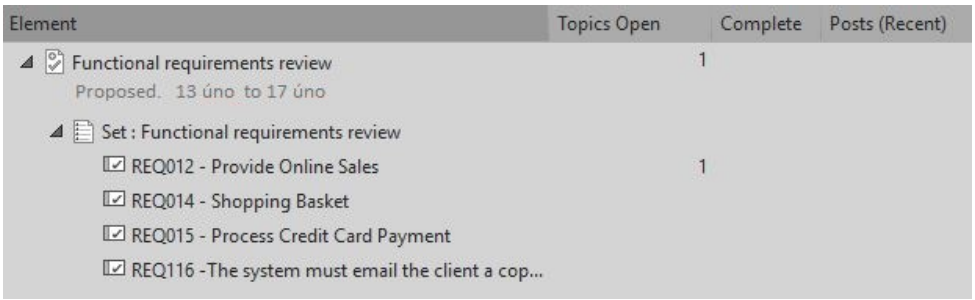
Vytváření specifikace požadavků je složitější na vypracování a není tak intuitivní jako například v softwaru Spira, ale následné provázaní a vysledovatelnost jednotlivých požadavků je velmi sofistikovaná.

Následně jsem zjišťoval jakou Enterprise architect nabízí podporu revizí pro jednotlivé artefakty. Software tuto funkčnost nabízí, avšak je tvořena jako diskuze k jednotlivým artefaktům, které v dané revizi jsou zohledněny. Vytvoření revize je jednoduché a nenabízí mnoho nastavení viz Obrázek 11

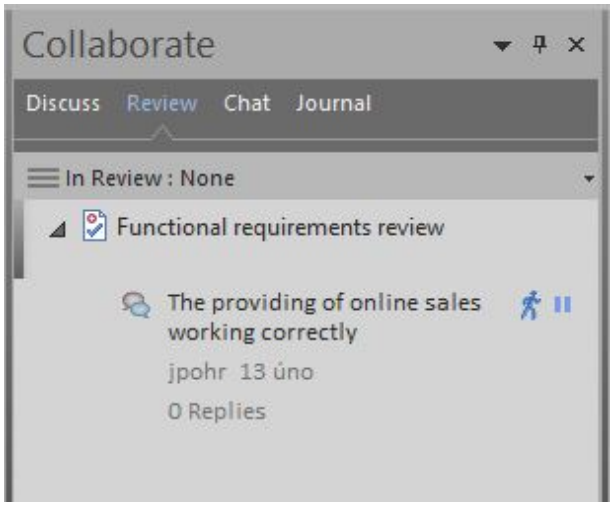


Obrázek 11: Vytváření revize

Následně se již do revize pouze přiřadí artefakty, které budou následně jejím předmětem. Ke každému artefaktu, který je obsažen v revizi mohou přispívat do diskuze všichni účastníci, kteří jsou součástí revize viz Obrázek 12 a 13.



Obrázek 12: Seznam artefaktů podléhající revizi



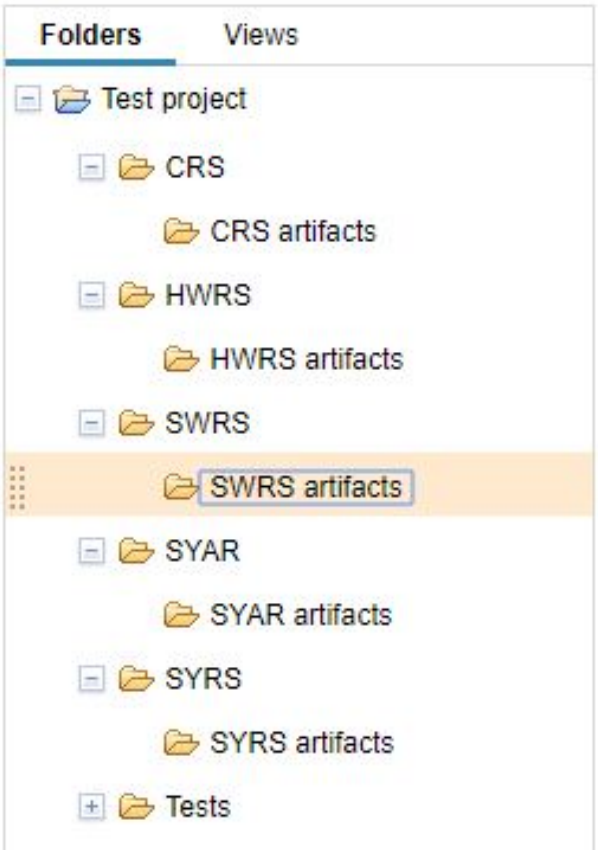
Obrázek 13: Diskuze ke konkrétnímu artefaktu

V systému jsem však nenalezl možnost nastavení, v jaké fázi revize se dané artefakty nacházejí. Cele revidování je tedy tvořeno pouze diskuzi, do které přispívají jednotliví účastníci. Ačkoliv je tento software velmi rozsáhlý nabízí pouze jednoduchou formu revizí, která pro mou diplomovou práci by dostačovala pouze částečně.

4.4 Rational DOORS Next Generation

Software který je spravován a vyvíjen společností IBM. Tento systém je rozsáhlý a je členěný do více nástrojů, které zpřístupňují funkčnosti pro specifickou část vývojového cyklu. Software nabízí 60 dní zkušební verzi, avšak tato verze musí být nainstalovaná na vlastní server. Nebo 45 denní zkušební verzi, která je spuštěna v cloudu IBM. Cena toho software začíná na 820$ za jednu uživatelskou licenci na jeden měsíc.

Při zkoumání, jaké funkce pro podporu revizí při vývoji software nabízí, jsem používal verzi, která byla nainstalována na vlastním serveru. Projekty jsou rozdělený do jednotlivých modulů, které představují jaké artefakty se v nich budou nacházet viz Obrázek 14.

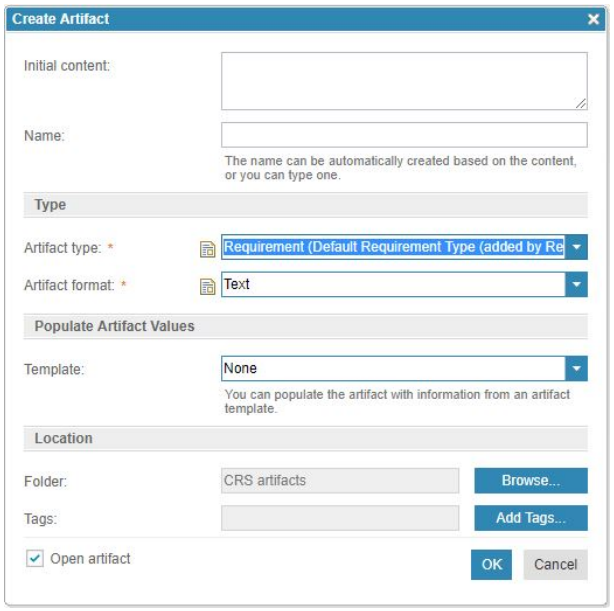


Obrázek 14: Rozdělení artefaktů do modulů

Tyto moduly jsou vytvářený uživatelem, který sám určuje jejich rozdělení a názvy.

- **CRS** - Zákaznický specifikace požadavků
- **HWRS** - Hardwarová specifikace požadavků
- **SWRS** - Softwarová specifikace požadavků
- **SYAR** - Požadavky systémové architektury
- **SYSR** - Systémová specifikace požadavků
- **Tests** - Požadavky týkající se testů

Do těchto modulů se následně pak přiřazují konkrétní specifikace požadavků nebo jiné artefakty. Vytvoření jednotlivých artefaktů je jednoduché a probíhá ve dvou krocích. V prvním kroku se vytvoří základní struktura artefaktu a přiřadí se do vytvořeného modulu viz Obrázek 15.

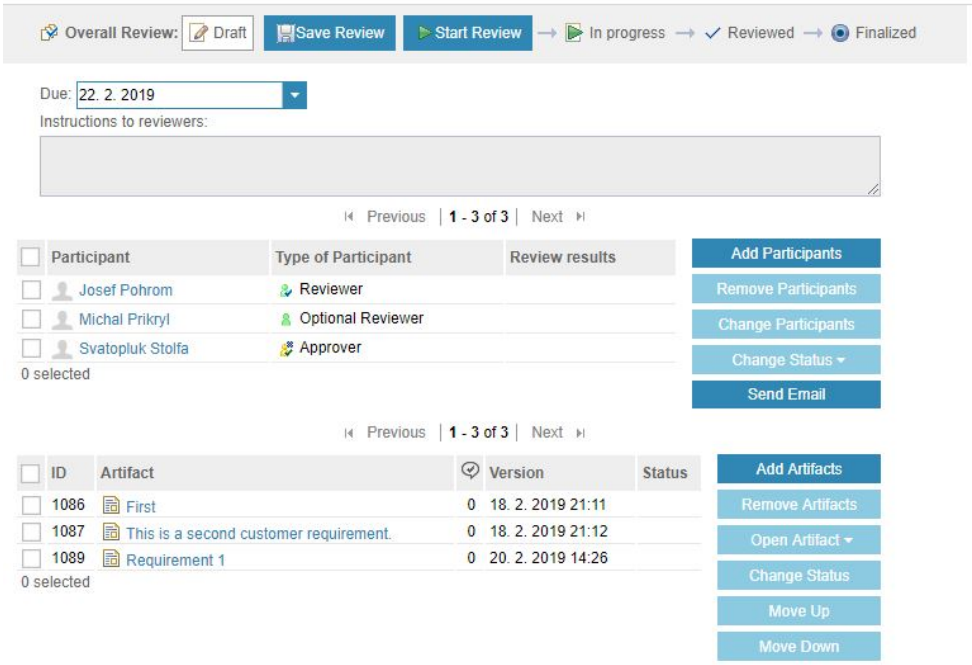


Obrázek 15: První krok vytvoření artefaktů

Po inicializaci artefaktu v tomto případě uživatelského požadavku se tento požadavek dále specifikuje. Přiřadí se v jakém stavu se požadavek nachází, o jaký typ požadavku se jedná a mnoho dalších nastavení.

Následně jsem se zajímal, jak lze jednotlivé artefakty revidovat a jak velkou flexibilitu software nabízí. Software nabízí podporu revidování, je zde možné zahrnout více účastníků vývoje, kteří mají přistup do systému Doors next generation. Při vytváření nové revize stačí zadat pouze její název a popřípadě popis. Následně se přidělí artefakty a přiřadí se členové, kteří se na daném revizi budou podílet a jejich role:

- **Reviewer** - Musí revidovat všechny položky v dané revizi.
- **Optional Reviewer** - Neovlivňují průběh revize.
- **Approver** - Schvaluje všechny revize artefaktu před jejím dokončením.



Obrázek 16: Detailní nastavení revize

Po spuštění celé revize všichni její účastníci přiřazují stavy k jednotlivým artefaktům. Mohou je označit jako revidované nebo revidované s komentářem, kde mohou zapsat nějakou poznámku, která se týká daného artefaktu. Popřípadě se také mohou zdržet revidování u daného artefaktu. Ze všech vykonaných revizí program umožňuje vygenerovat report, který obsahuje všechny účastníky, kteří se podíleli na revidování, veškeré artefakty a popřípadě komentáře, které účastníci zaznamenali během revidování.

Tento software splňuje z větší části požadavky, které jsou předmětem celé diplomové práce, ale flexibilní vytváření jednotlivých formulářů pro revizi neobsahuje.

4.5 Zhodnocení

Mnou zkoumané nástroje zpřístupňují poměrně dobrou funkcionalitu správy a řízení vývoje projektů. Čím rozsáhlejší funkcionalitu však software nabízel, tím složitější jeho ovládání a nastavování bylo. Cenová relace těchto softwaru se pohybovala mezi 200$ až 820$.

Nativní podporu review z těchto testovaných softwaru měly jen dva a to Enterprise architect a Rational DOORS Next Generation. Spira neobsahovala žádnou formu review a software ReqView měl pouze možnost přidávat komentáře k jednotlivým artefaktů, což by se dalo považovat za částečnou formu revidování. ReqView však vynikalo ve vytváření vlastních dokumentů, které mohly sloužit k revidování, ale provázat jednotlivé artefakty z jiných dokumentů s dokumentem, který jsem sám definoval jsem nedokázal.

Přiřazování rolí k jednotlivým revizím nabízel pouze software Rational DOORS Next Generation a tyto role již měl předdefinované. Ostatní testované softwary tuto funkcionalitu neměly ani jsem nenalezl její nastavení.

Při testování těchto softwaru jsem se tedy především zaměřil na tuto funkcionalitu:

a) Nativní podpora review

b) Vlastní definovatelnost review dokumentu

c) Role v review

d) Jednoduchost ovládání

Body a až c jsem zaznamenal do tabulky jako ANO/NE a jednoduchost ovládání známkou jako ve škole viz Tabulka 7.

Software	a	b	c	d
Spira	NE	NE	NE	1
ReqView	NE	ANO	NE	3
Enterprise architect	ANO	NE	NE	3-
DOORS Next Gen.	ANO	NE	ANO	2

Tabulka 7: Srovnání funkcionalit jednotlivých softwaru

Z otestovaných nástrojů se jeví jako nejlepší nástroj od IBM Rational DOORS Next Generation, který splňuje nejvíce kriterii. ReqView a Enterprise architect splňovaly pouze jedno z hlavních kriterií, avšak co se týče revidovaní artefaktů bych upřednostnil Enterprise architect. Spira sice nesplnila žádné z kriterií, avšak její ovládání bych hodnotil velmi kladně a tento nástroj bych doporučil především vývojářům v malých týmech.

5 Vlastní řešení

Po průzkumu dostupných nástrojů viz kapitola 4. bylo zjištěno, že nástroje sice podporují revidování jednotlivých artefaktů vývoje, nicméně styl revidování jednotlivých artefaktů nekorespondoval se stylem, který je běžně využíván v praxi dle Automotive SPICE.

5.1 Vize

Nástroj bude sloužit pro podporu a záznam revizí při vývoji softwaru. Nástroj bude možno definovat různé formuláře pro jednotlivé revize. Bude umět importovat artefakty z nástroje Rational DOORS Next Generation, který je popsán v kapitole 4.4. Jednotlivé revizní formuláře budou tedy členěny k jednotlivým pracovním produktům, které si uživatel bude tvořit sám. Celý systém bude tvořen jako webová aplikace.

5.1.1 Funkcionalita systému

1. První základní funkcionalitou systému bude definování vlastních formulářů pro jednotlivé revize. Jako předlohu pro formuláře jsem dostal k dispozici excelový soubor, ve kterém můj vedoucí tvořil a uchovával jednotlivé formuláře pro různé revize. Příklad formuláře, který jsem měl k dispozici viz Obrázek 17.

Artifact ID	Column 2	Column 4	Responsible person	Column 5	Column 6	Column 7	Approval	Column 9	Column 10
1			Jan Novák				Approved		
2			Jan Novák				Approved		
3			Jan Novák				Approved		
4			Jan Novák				Approved		
5			Jan Novák				Approved		
6			Jan Novák				Approved		
7			Jan Novák				Approved		
8			Jan Novák				Approved		
9			Jan Novák				Approved		
10			Jan Novák				Approved		
11			Jan Novák				Approved		

Obrázek 17: Předloha revizního formuláře

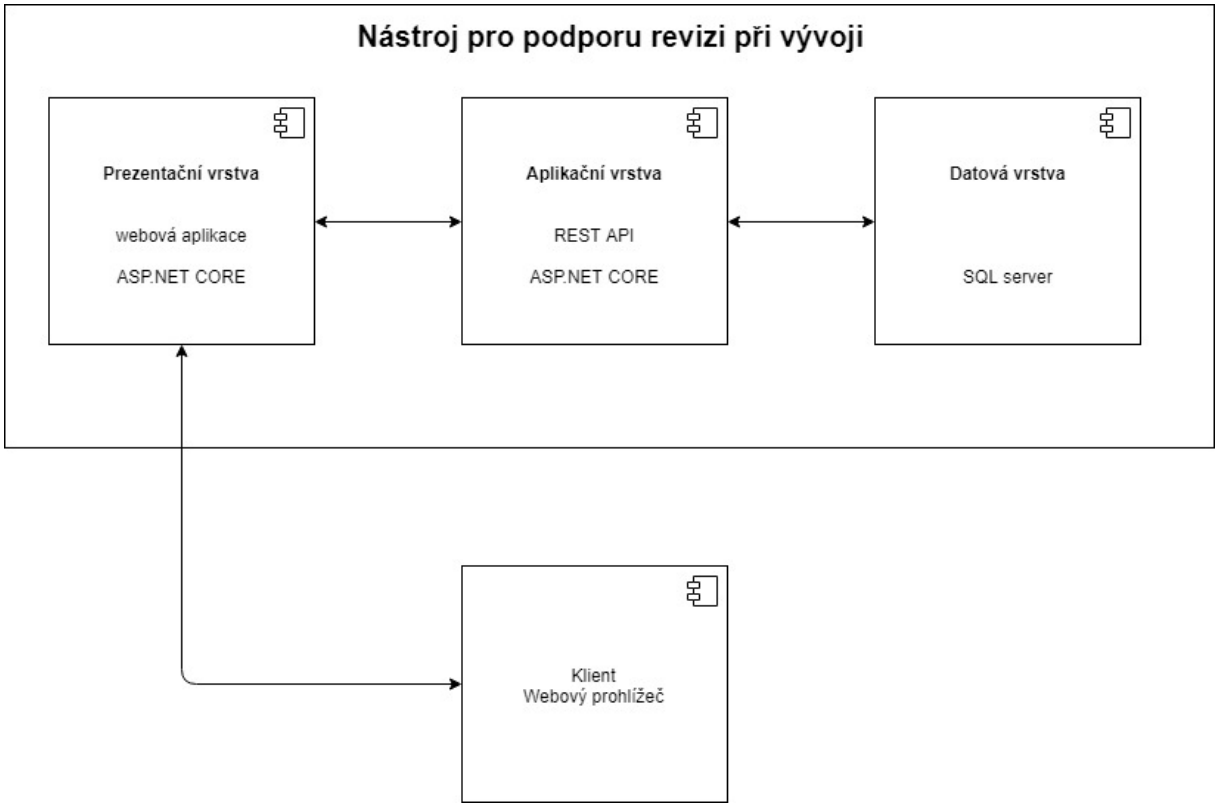
   Tato předloha musela být značně anonymizována a slouží pouze jako ukázka, jaké formuláře mohou být použity v praxi při prováděných revizích, které jsou vyžadovány dle standardu Automotive SPICE. Tento styl formulářů se budu snažit napodobit i v aplikaci.

2. Import artefaktů z nástroje Rational DOORS Next Generation, který slouží pro management požadavků. Artefakty, které podléhaly revizi, byly přepisovány ručně do formulářů, které jsem zmínil v předchozím bodě. Tento postup by měl práci zefektivnit.

5.2 Architektura nástroje

Po analýze hotových řešení a požadavků vedoucího se jevily dvě možnosti, jakou architekturu zvolit. Jednou z možností byla desktopová aplikace a druhou možností webová aplikace (klient-server). Zvolil jsem webovou aplikaci, protože se její výhody, kterými jsou odpadnutí instalace, nezávislost na operačním systému uživatele a jednoduší nasazení aktualizace během vývoje nástroje zdály být nejvýhodnějšími.

Celá aplikace je rozdělena do jednotlivých logických komponent, které obstarávají jednotlivé funkcionality celého systému. Tyto komponenty jsou celkem tři viz Obrázek 18.

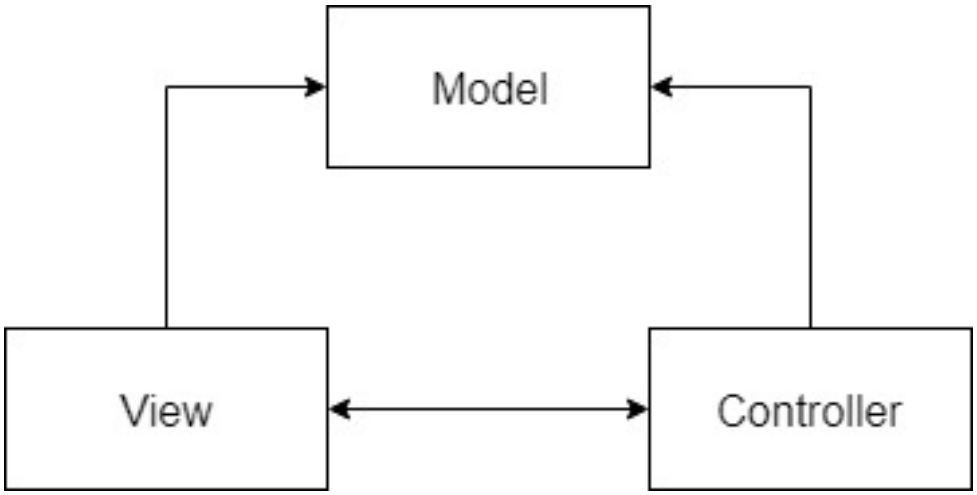


Obrázek 18: Diagram komponent rozložení celého systému

5.2.1 Prezentační vrstva

Prezentační vrstva obstarává komunikaci mezi uživatelem a aplikační vrstvou. Pro vývoj prezentační vrstvy aplikace, jsem využil technologii ASP.NET CORE za použití MVC architektury a programovacího jazyka C#. Prezentační vrstva zasílá jednotlivé požadavky o data, které uživatel požaduje, na REST API, které je součástí aplikační vrstvy. Důvodem oddělení prezentační vrstvy a aplikační vrstvy je následná větší flexibilita při úpravách uživatelského rozhraní, které v tuto chvíli nevyužívá žádnou moderní technologii, která by byla zaměřena převážně na tvorbu uživatelského rozhraní jako například React, Angular nebo jiné.

- **Model View Controller (MVC)**
  Architektura, která odděluj logiku od výstupu, který je generován uživateli. To značně zpřehledňuje programátorský kód, protože jednotlivé třídy do sebe nemíchají například HTML značení s logikou, která vytváří data jež se budou zobrazovat na výstupu. Jak již název napovídá tato architektura je rozdělena do tří částí (komponent). Na Obrázku 19. lze vidět jak jsou tyto komponenty mezi sebou propojeny.[9]



Obrázek 19: Diagram MVC architektury

1. **Model -** je objekt, který představuje nějaké informace o doméně do které patří. Model je nevizuální objekt obsahuje pouze data a logiku, která se však netýká uživatelského rozhraní (pracuje pouze s daty, které obsahuje nebo mu jsou dány k dispozici).[9]
2. **View -** obstarává zobrazení dat, které jsou obsaženy v modelu, který je do view předán. Žádnou jinou práci s daty, které model obsahuje nevykonává. Další činnost, kterou view obstarává je zasílání požadavků na kontroler o nové data.[9]
3. **Controller -** zpracovává uživatelský vstup, který přijde z view a následně pracuje s modely, které jsou nezbytné k vrácení požadovaného výstup do uživatelského rozhraní (view).[9]

Jak jsem již na začátku zmínil, pro vývoj jsem využíval programovací jazyk C#, následující ukázky kódu pro kontroler a model jsou tedy vytvořeny pomocí jeho syntaxe. Ukázka kódu viz Výpis 1.

```
public class ProjectController : Controller
{
   public IActionResult ShowProject()
    {
      ProjectViewModel project = new ProjectViewModel();
      ViewBag.Project = project;
      return View("Projects");
    }
}
```

Výpis 1: Ukázka zdrojového kódu kontroleru v jazyce C#

Z výpisu kódu je jasné, že kontroler v jazyce C# je tvořen třídou, která dědí z rodičovské třídy *Controller*. Tento kontroler má v sobě jednu funkci (metodu) s názve *ShowProject* jejíž návratová hodnota je *IActionResult*. Tato metoda vztv859 objekt *ProjectViewModel*, který ponese nějaká data týkající se projektu. Tento objekt je následně předán do view pomocí vlastnosti *ViewBag*, která má možnost v sobě vytvářet dynamicky proměnné. Funkce nakonec vrátí view s názvem Projects.

Model, který ke použit v kontroléru ve Výpisu 1 je zobrazen ve Výpisu 2.

```
public class ProjectViewModel
{
   public int Id { get; set; }
   public string Name { get; set; }
   public string Description { get; set; }
}
```

Výpis 2: Ukázka zdrojového kódu modelu v jazyce C#

Jak je vidět z Výpisu 2. tak model je pouze třída, která má vlastnosti, které jsou zobrazeny nebo použity ve view. Může také obsahovat funkcionalitu (metody), ta však ve Výpisu 2 není zobrazena.

- **Razor View engine**
  Razor je syntaxe značení, která slouží pro vkládání logiky kódu do webových stránek, které jsou pak zpracovány na serveru a je z nich vygenerováno view, které se zobrazí uživateli. Syntaxe se skládá z HTML značení a C# syntaxe. Soubory do kterých je kód s touto syntaxí vkládán má obvykle příponu *.cshtml*. HTML značení v této syntaxi není nijak odlišné od klasické HTML syntaxe. Pokud však je nutno využít C# kód je nutné využít symbol @ aby server věděl, že má vykonat naprogramovanou funkcionalitu. [11] Příklad kódu lze vidět na Výpisu 3.

```
<table>
   <tr><td>Project id: </td><td>@ViewBag.Project.Id</td></tr>
   <tr><td>Project name: </td><td>@ViewBag.Project.Name</td></tr>
   <tr><td>Project description: </td><td>@ViewBag.Project.Description</td>
      </tr>
</table>
@for(int i = 0; i < 3; i++)
{
   <div>@i</div>
}
```

Výpis 3: Ukázka zdrojového kódu Razor syntaxe

Tento kód vygeneruje view, které obsahuje tabulku naplněnou daty o projektu, které byly předány v kontroléru viz Výpis 1. A to za pomoci vlastnosti ViewBag, která je tvořená dynamicky za běhu podrobnější popis ViewBag lze nalézt zde [10]. V další části jsou vygenerovány tři div elementy, které obsahují hodnotu *i*, která se inkrementuje při průchodu cyklem for.

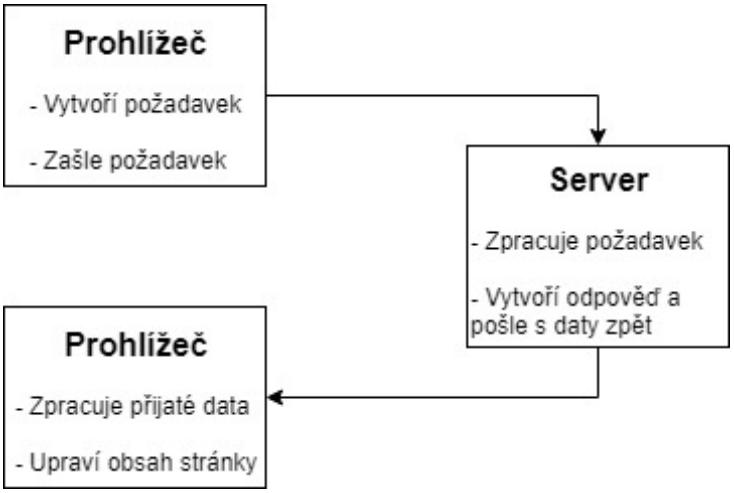
- **AJAX - Asynchronous JavaScript and XML**
  Jelikož v mé aplikaci je nutné tvořit formuláře dynamicky a dynamické tvoření formulářů za pomoci razor syntaxe, kterou jsem zmínil výše, by bylo obtížné, využil jsem programovací jazyk javascript, který zpřístupňuje tuto funkcionalitu.

  Je to technologie, která dokáže zasílat požadavky na server společně s daty, které obsahuje například formulář a také přijímat odpovědi s daty ze serveru. Data mohou být posílány v různých datových formátech jako je například XML nebo JSON. Důležitá vlastnost této technologie je, že dokáže komunikovat se serverem asynchronně to znamená, že stránka nemusí být obnovena při zaslání dotazu na server. [12]

```
$.ajax({
       type: 'POST',
       url: "/Project/ShowProject",
       dataType: 'json',
       contentType: "application/json; charset=utf-8",
       data: data,
       success: function (result) {
           //functionality after success
       }
   });
```

Výpis 4: Funkce která zasílá požadavek s daty

Jedná z možnosti, jak zaslat požadavek pomocí AJAX lze vidět na Výpisu 4. Celá komunikace AJAXu se serverem je zachycena na Obrázku 20.



Obrázek 20: Diagram komunikace AJAX se serverem

5.2.2 Aplikační vrstva

Aplikační vrstva slouží jako propojení mezi prezentační vrstvou a datovou vrstvou. Aplikační vrstva je realizována jako REST API a využívá technologie ASP.NET Core a programovacího jazyka C#. Pro práci s databází využívá Entity framework, který je skvělým nástrojem pro namapování databázového modelu do aplikace a následné dotazování databáze o data.

- **REST - Representational State Transfer**
  Je to architektonický styl, který slouží pro distribuované hypermediální systémy. Tento styl má 6 základních principů, které musí být dodrženy, aby bylo možno tvrdit, že je RESTful.

  1. **Klient-server -** Tento princip umožňuje oddělit logiku ukládání a zpracování dat od uživatelského rozhraní. Tímto principem tak zlepšíme přenositelnost uživatelského rozhraní mezi více platforem. Také je usnadněna škálovatelnost systému pří jeho dalším vývoji, protože komponenty (uživatelské rozhraní a logika běžící na serveru) mohou být vyvíjeny nezávisle na sobě.[13]
  2. **Bezstavovost -** Tento princip říká, že veškeré informace, které jsou potřebné k úspěšnému identifikování požadavku, který je zaslán na server, musí být obsaženy v tomto požadavku. Tudíž nesmí těžit z jakéhokoliv kontextu, který je na serveru uložen. Avšak stav relace je výhradně uchováván na straně klienta. Nevýhodou však je možnost sníženi výkonnosti sítě, z důvodů zasílání opakujících se dat na server, jelikož data z předchozích požadavků nemohou být na serveru ponechány ve sdíleném kontextu.[13]
  3. **Cacheable -** Navazuje na předešlý princip a to tak, že se snaží vylepšit efektivitu sítě a to tak, že data která jsou odesílána jako odpověď na klientské dotazy, byl

značeny jako „cacheable“ nebo „non-cacheable“. Pokud je odpověď označena jako „cacheable“, tak uživatelská mezipaměť má právo znovu použít data, pro ekvivalentní dotazy, které zašle opětovně později.[13]

4. **Jednotné rozhraní -** Je kladen důraz na jednotné rozhraní mezi komponentami. Tím se zjednodušuje celková systémová architektura a viditelnost jednotlivých interakcí je vylepšena. Aby této vlastnosti bylo dosaženo jsou definována čtyři omezení pro rozhraní. Identifikace zdrojů, manipulace se zdroji skrz reprezentace, zprávy popisují samy sebe, hypermedia jak engine aplikačního stavu.[13]
5. **Vrstvený systém -** Umožňuje architektuře, aby byla rozložena do vrstev tím je omezeno chování komponent, které nemůžou vidět mimo vrstvu se kterou spolupracuje. Jelikož je omezena viditelnost těchto komponent je lépe definovatelná komplexnost celého systému. Tyto vrstvy mohou být využity k zapouzdření starých služeb a k ochraně nových služeb před starými klienty. [13]
6. **Kód na vyžádání -** Dovoluje rozšířit funkcionalitu klienta tím, že je možné stažení a spuštění kódu ve formě appletu nebo skriptu. Toto usnadňuje a snižuje počet funkcí, které mají být předem naprogramovány. Toto povolení sice zlepšuje rozšiřitelnost systému, avšak snižuje viditelnost a proto je tato možnost pouze volitelná jestli bude umožněna.[13]

Spojení REST architektury společně s technologii ASP.NET Core tvoří skvělou kombinaci pro vytvoření REST API. REST API je tvořena opět z kontrolérů, které pomocí funkcí zpřístupňuje rozhraní pro uživatele. Podle návratové hodnoty funkce je vracen požadovaný výstup. REST kontrolér vytvořený pomocí ASP.NET Core technologie a jazyka C# by vypadal viz Výpis 5.

```
[Route("api/Project")]
public class ProjectController : Controller
{
   [HttpPost]
   [Route("ShowProject")]
   public string ShowProject([FromBody]string name)
   {
      return name;
   }
}
```

Výpis 5: Ukázka zdrojového kódu REST kontroleru v jazyce C#

Z výpisu kódu jde vidět, že REST kontrolér se od MVC kontroléru, který je na Výpisu 1. se příliš neliší, přibyly zde však notace, které jsou nezbytné pro jeho správné fungování jako API. Tyto notace jsou v hranatých závorkách a upřesňují konfiguraci daného kontroléru.

– Notace ***Route*** určuje URL adresu, na které je daný kontrolér dostupný. V tomto případě na */api/Project* a jeho funkce *ShowProject* je tedy dostupna na */api/Project/ShowProject*. Tímto způsobem je přesně definováno, kde se daná funkcionalita nachází.

– Další notace je **FromBody** ta se váže k parametru funkce *string name* a říká, že tato funkce očekává na vstup řetěze, který obsahuje jméno. Tato hodnota přijde s požadavkem od uživatele.

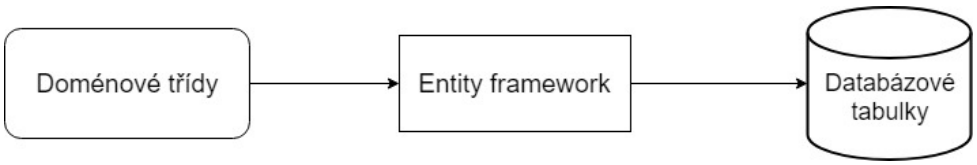
– Poslední notace je **HttpPost** a ta určuje jaka HTTP metoda při zasílání požadavku musí být použita pro přístup k této funkcionalitě. Existují ještě další tři notace, kterými jsou **HttpGet, HttpPut, HttpDelete**.

- **Entity framework Core**

  Entity framework Core je open source technologie, která je zároveň multiplatformní a vychází z technologie Entity framework, která slouží pro přístup k datům. Entity framework Core slouží k objektově relačnímu mapování a umožňuje vývojářům jednoduše pracovat s databází, přičemž mohou používat objekty .NET technologie. Také odbourává nutnost vývojářům psát kód, který by sloužil k přístupu k datům v databázi.[14]

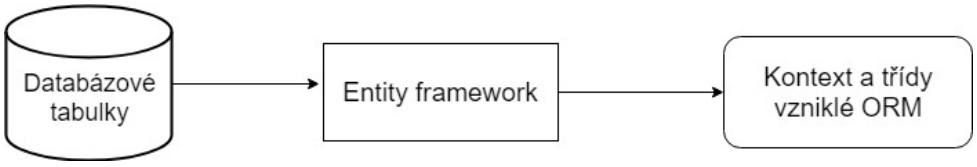
  Entity framework Core definuje dvě možnosti přístupu k databázi, těmi jsou

  1. **První kód -** tento přístup spočívá v tom, že nejdříve se vytvoří třídy a vazby mezi nimi pomocí programovacího jazyka a ty se následně namapují do databáze. Podrobnější postup jak použít tuto metodu viz [15]. Celý postup vytvoření databáze z doménového modelu lze vidět na Obrázku 21.



Obrázek 21: Diagram postupu generování databáze z doménového modelu pomoci Entity framework Core

  2. **První databáze -** tento přístup spočívá v tom, že databázové tabulky a jejich vazby jsou namodelovány pomocí jazyka SQL a nahrány do databáze. Entity framework Core je schopen na základě modelu z databáze vytvořit objektové relační mapování a tím tak vytvořit jednotlivé třídy, které lze následně používat při vyvíjení kódu. Toto mapování se provede příkazem:
  Scaffold-DbContext "Server=[adresa databáze];Trusted_Connection=True;"
  Microsoft.EntityFrameworkCore.SqlServer -OutputDir [název složky]
  a je také zobrazeno na Obrázku 22.



Obrázek 22: Diagram postupu generování tříd z databáze pomoci Entity framework Core

Tento postup jsem také zvolil při svém vývoji aplikace. Podrobnější návod, jak postupovat viz [16]

Pro práci s databází je nutno mít vytvořenou instanci kontextové třídy, které umožňuje zasílat dotazy na databázi. V mé aplikaci je tato třída vkládána pro použití pomocí návrhového vzoru dependency injection. Následné dotazy na databázi jsou tvořeny pomocí LINQ. Kód, který je na Výpisu 6 ukazuje jednoduchý dotaz na databázi.

```
[Route("api/Project")]
public class ProjectController : Controller
{
   [HttpGet]
   [Route("ProjectDetail")]
   public Project ProjectDetail(int id)
   {
      Project p = context.Project.Where(x => x.projectId == id).
         FirstOrDefault();
      return p;
   }
}
```

Výpis 6: Ukázka dotazu pomocí Entity framework Core

Tento dotaz dostane z databáze informace o projektu jehož požadované id přišlo od uživatele na API. Následně vytvoří instanci objektu s tímto záznamem a vrátí ho uživateli.

- **JWT - JSON Web Token**
  Je to otevřený standard, který definuje bezpečný přenos informací mezi zúčastněnými stranami a to za pomocí JSON objektu. Informace v tokenu jsou důvěryhodné, protože je digitálně podepsán. Existují dva druhy podpisu buď jeho zakódování pomocí algoritmu HMAC ne veřejným a privátním klíčem.[17]

  JSON Web Token se používá ve dvou případech

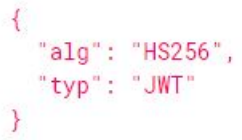
  1. **Autorizace -** Když je token přiřazen uživateli, tak k veškerým požadavkům, které uživatel bude posílat na server, bude tento token přiložen. To umožní uživateli při-

stupovat k adresám serveru, které jsou zabezpečeny proti přístupu uživatelů bez toho tokenu. [17]

2. **Výměna informací -** Jelikož tento token může být zabezpečen veřejným a privátním klíčem, je zajištěno, že identita uživatele, který posílá data bude ověřena.[17]

Struktura tokenu je tvořena ze tří částí, které jsou odděleny tečkou.

- **Hlavička -** obvykle obsahuje dvě části. Jedna část obsahuje o jaký druh tokenu se jedná a druhá část jaký algoritmus byl využit k podepsání tokenu. [17] Hlavička tokenu je na Obrázku 23.



Obrázek 23: Hlavička JWT tokenu

- **Payload (data) -** v téhle části tokenu jsou obsaženy tvrzení (claims), které se týkají nejčastěji uživatele. Jsou celkem tři druhy těchto tvrzení. [17] Data v tokenu viz Obrázek 24.
  - **Registrované -** Tyto tvrzení jsou předdefinované avšak nemusejí se používat.
  - **Veřejné -** Mohou být definovány podle uživatelů, kteří JWT používají.
  - **Privátní -** Vlastní tvrzení vytvořené za účelem šíření informací, na kterých se obě strany shodnou.

Obrázek 24: Payload (data) v JWT tokenu

- **Podpis -** K vytvoření podpisu je třeba mít vytvořeny předešlé dvě části tokenu (Hlavičku a Payload (data)), algoritmus, který je využit a tajnou sekvenci znaků.[17] Vygenerování podpisu lze vidět na Obrázku 25.

```
HMACSHA256(
  base64UrlEncode(header) + "." +
  base64UrlEncode(payload),
  tajna sekvence znaku
)
```

Obrázek 25: Podpis JWT tokenu

Celou strukturu vygenerovaného tokenu pak tvoří sekvence znaků. Konkrétní sekvence, která by vznikala z předchozích bodů by vypadala viz Obrázek 26.

```
eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.
eyJzdWIiOiIxMjM0NTY3ODkwIiwibmFtZSI6Ikph
biBOb3ZhayIsImVtYWlsIjoiamFubm92YWtAdnNi
LmN6In0.
eWgJlgnUAjGoyG54mQhCkhZBViCcP25-
MJCiXsKO5kY
```

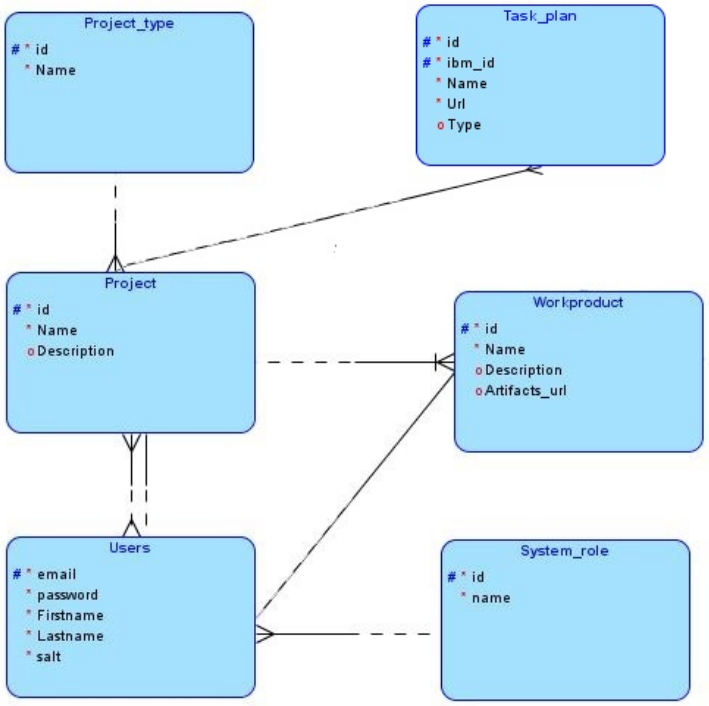
Obrázek 26: Finální struktura JWT tokenu

5.2.3 Datová vrstva

Datová vrstva je perzistentní a slouží k uchování veškerých informací, jak o uživatelích, tak o projektech a jejich pracovních produktech, revizních formulářích a konkrétních revizích, které uživatel vytvoří. Datová vrstva používá technologii SQL server, kde databáze je tvořena relačním datovým modelem.

Aby jednotlivé popisy databáze byly přehlednější, rozdělil jsem jej pro tyto účely na tři části. Pro zobrazení jednotlivých entit v databázi používám konceptuální model.

1. **Uživatelská část -** Tato část uchovává informace týkající se uživatelů, jakou roli v systému mají a k jakým projektům a pracovním produktům jsou přiřazeni. Toto schema je na Obrázku 27.

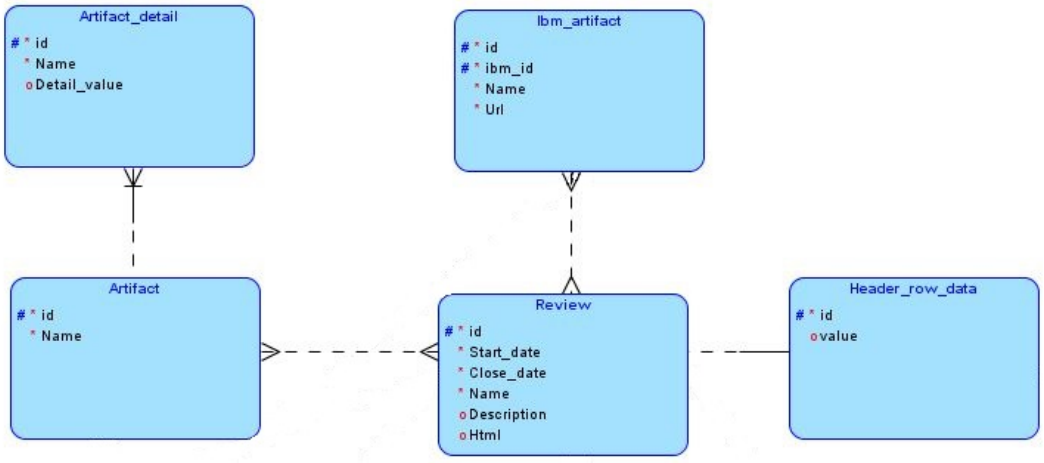


Obrázek 27: Část databáze, která ukládá data o uživatelích

Další vazby, které tyto entity obsahují s entitami, které na obrázku nejsou, jsem pro přehlednost smazal, nicméně lze je vidět na Obrázku 30., kde je vyobrazeno celé schéma databáze. Entita *Project_type* obsahuje typy projektu, které jsou na výběr, podle této hodnoty je následně generováno specifické view pro uživatele, které se liší možností importováním artefaktů, které budou obsaženy v revizi. *System_role* je entita, která obsahuje typy uživatelských rolích v systému, nicméně v systému v tuto chvíli nejsou tyto role implementovány, protože prozatím nebylo potřeba rozlišovat práva uživatelů. *Task_plan* slouží k napojení na plánovací vrstvu a jsou zde ukládány data ze sestav. Entita *Project* obsahuje jednotlivé *Workproduct*, pro které pak následně vznikají revize. Ty jsou popsány v následujícím bodě.

2. **Revizní část -** Tato část databáze ukládá jednotlivé artefakty, které jsou importovány z platformy Rational DOORS Next Generation. Tyto data jsou ukládány do tabulky *IBM_artifact*. Dále jsou v databázi připraveny tabulky pro libovolné artefakty, které mohou mít definované také libovolné vlastnosti. K tomuto slouží tabulky *Artifact* a *Artifact_detail*. Tyto tabulky jsou vázány k samotné revizi ve vztahu M:N avšak tento vztah není povinný, jelikož jednotlivé Revize nemusí obsahovat žádné artefakty, protože jedním z úkolů bylo vytvořit formulář pro nespecifikovanou revizi. Poslední tabulka je *Header_row_data* v této tabulce jsou ukládaný hodnoty, které jsou při provádění revize

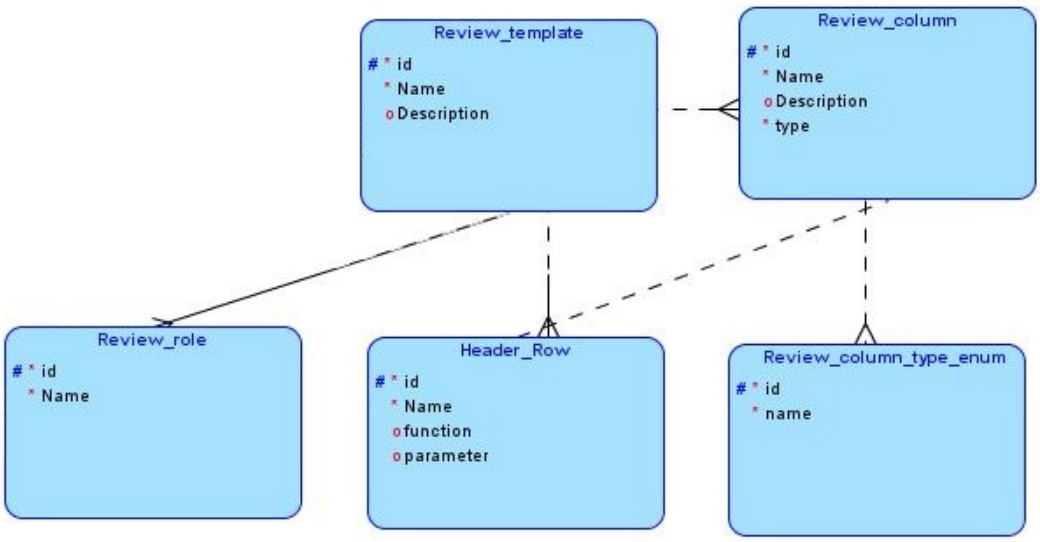
vypočteny nebo zapsány v závislosti na šabloně, která je pro revizi zvolena. Model teto části databáze viz Obrázek28.



Obrázek 28: Část databáze, která ukládá data artefaktech a revizích

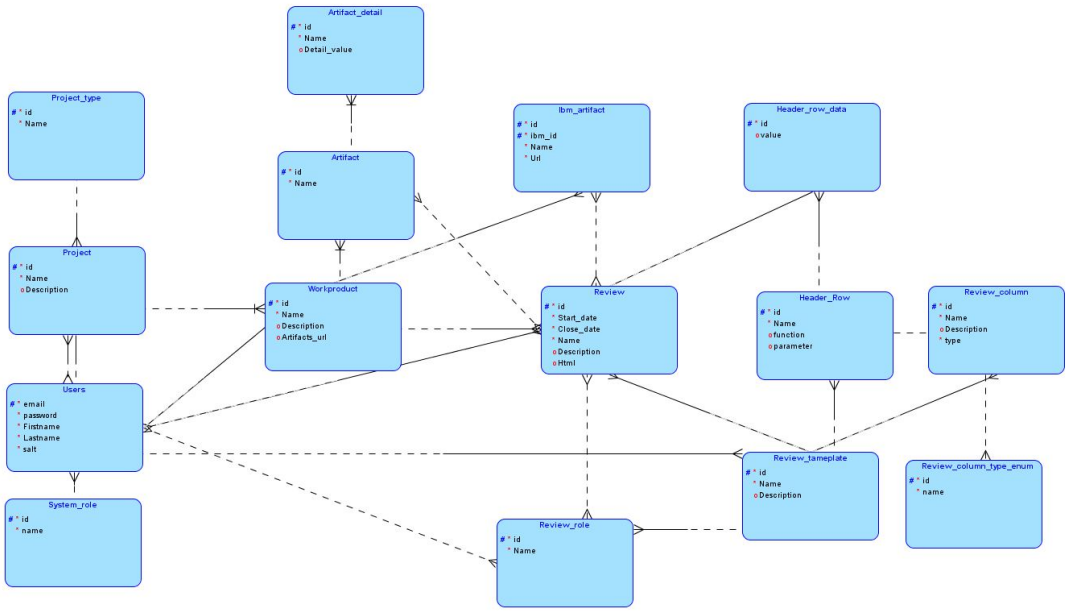
Tabulky s informacemi o jednotlivých artefaktech jsou ještě ve vztahu s tabulkou *Work-product*, která je zmíněna v předchozím bodě. Tento vztah však jde vidět až na Obrázku 30.

3. **Šablonová část -** V této části databáze jsou data, které slouží k zpětnému vygenerování šablony, kterou vytvoří uživatel. Tabulka *Review_column* slouží k uložení jednoho sloupce v šabloně pro revizi, v této tabulce je definováno o jaký typ sloupce se jedná (jak bude vyplněn).V aplikaci jsou prozatím dvě možnosti. Jednou z možností je čistý text a druhou je vyběr z možností (dropdown buuton). Pokud se jedná o sloupec, který umožňuje z výběru možností, tak tyto možnosti jsou uloženy v tabulce *Review_column_type_enum*. Poslední položkou, kterou lze definovat v šabloně pro revizi, jsou role jednotlivých účastníků revize. Ty jsou následně uloženy do tabulky *Review_role*. Tabulka *Review_template* udržuje vazby mezi jednotlivými sloupci. Model této části databáze je na Obrázku 29.



Obrázek 29: Část databáze, která ukládá data pro vytvoření šablony revize

Role, které jsou definovány, jsou následně přiřazeny jednotlivým uživatelům. Tato vazba je vidět až na Obrázku 30., který obsahuje kompletní schéma databáze.



Obrázek 30: Kompletní schema databáze

5.3 Stahování dat z Rational DOORS Next Generation

Jelikož můj vedoucí této práce využívá nástroj Rational DOORS Next Generation, pro správu požadavků a dalších artefaktů vývoje, měl jsem za úkol zjistit možnosti napojení na tento nástroj

a stažení dat z něj.

Zvažoval jsem výběr z celkem 3 možností.

1. **Reportable api -** Rational Requirements Composer poskytuje REST API, která dokáže zpřístupnit informace vztahující se k artefaktům požadavků. Tento způsob získávání dat byl poněkud složitý, proto jsem od tohoto řešeni odstoupil. Dokumentace, podle které jsem se snažil data ze serveru získat viz [18].

2. **OSLC - Open Services for Lifecycle Collaboration -** Rational DOORS Next Generation umožňuje práci s jeho daty za použití toho standartu, který definuje soubor specifikací jež umožňuje integraci se softwarem.[19]

   Podrobnější návod, jak tento standart využít pro integrací s aplikací Rational DOORS Next Generation viz [20]. Nicméně ani tento standart jsem nevyužil pro získávání dat z aplikace.

3. **Sestavy -** Rational DOORS Next Generation obsahuje plugin, který dokáže generovat uživatelem specifikovanou sestavu artefaktů. Tuto možnost jsem také využil ve své aplikaci, protože její implementace byla nejjednodušší. Sestavu s daty si uživatel definuje sám a po jejím vytvoření vznikne URL adresa, která tyto data zpřístupní. Data na této adrese jsou popsány v XML. V mé aplikaci pak toto XML pouze zpracuji a zpracovaná data uložím do databáze.

   Příklad artefaktu, který je obsažen na vygenerované url adrese viz Výpis 7.

```
<results>
   <result>
      <PROJECT_NAME>Test project</PROJECT_NAME>
      <REFERENCE_ID>1088</REFERENCE_ID>
      <URL1_title>Test software requirement</URL1_title>
      <URL1>
      https://158.196.141.113/rm/resources/
         MB_a6bac0f46827445e8f5c51e8a5552a4e
      </URL1>
      <REQUIREMENT_TYPE>Requirement</REQUIREMENT_TYPE>
      <NAME>SWRS</NAME>
   </result>
</results>
```

Výpis 7: Ukázka artefaktu popsaného v XML

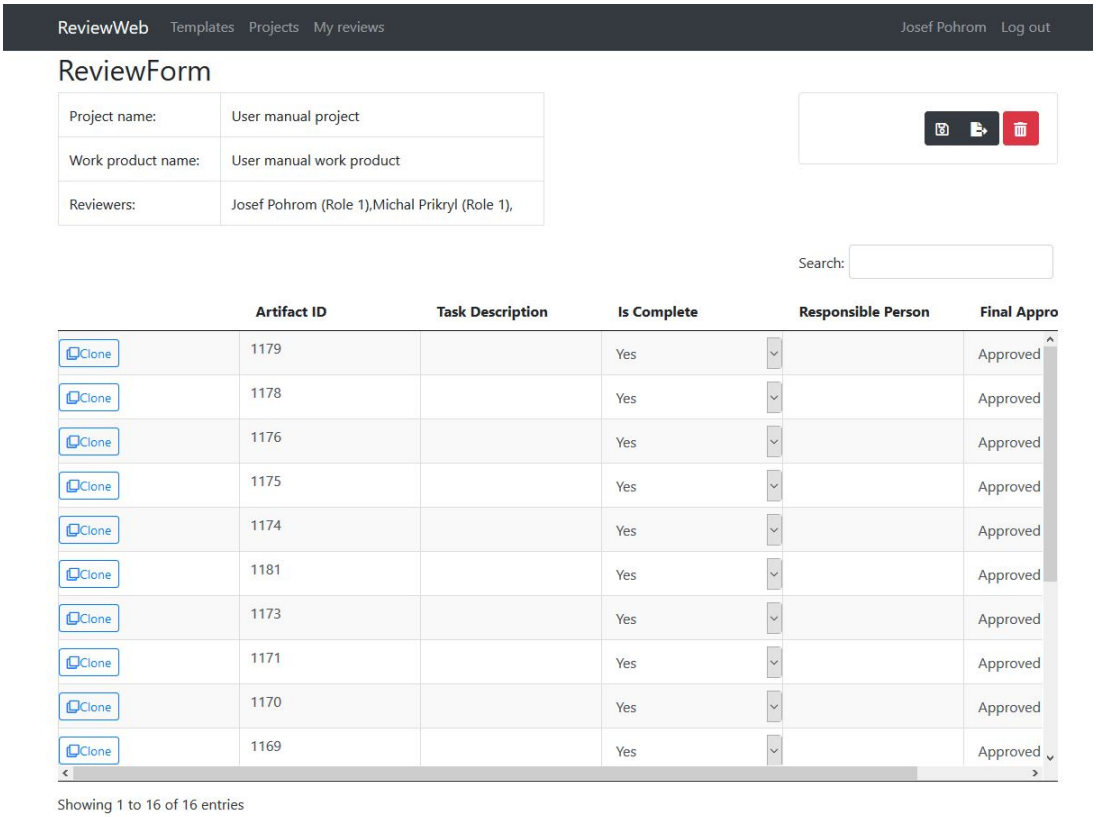
Toto XML má 4 parametry, které jsou vždy vygenerovány a ostatní parametry jsou poté volitelné při definování sestavy. Tyto 4 parametry jsou *PROJECT_NAME*,*REFERENCE_ID*,

URL1_title a *URL1*. Tyto parametry pak následně ukládám do databáze a ostatní parametry, které jsou volitelné ignoruji, protože url, která je obsažena v tomto XML odkazuje na konkrétní artefakt v systému, kde se dají dohledat ostatní detaily.

5.4 Ukázka aplikace

Revizní formuláře, které jsem obdržel od vedoucího práce a jsou využívány v praxi byly tvořeny v excelu, proto jsem se rozhodl tento styl ponechat. Veškeré formuláře, které jsou tedy tvořeny uživatelem vytváří HTML tabulkové prvky (*table*, *tr*, *th*, *td*). Tyto formuláře jsou navázány na knihovnu DataTables, která tyto formuláře rozšiřuje o funkcionalitu vyhledávání a řazení.

Při generování tohoto formuláře, uživatel zvolí jakými daty bude formulář vyplněn popřípadě formulář nemusí být vyplněn žádnými daty v případě, že by uživatel žádná data nepotřeboval. Jak by vygenerovaný formulář vypadal lze vidět na na Obrázku 31.



Obrázek 31: Vygenerovaný revizní formulář s daty

Jak vytvořit takovýto formulář popisuji v uživatelském návodu, který je součásti příloh.

5.5 Návrhy ke zlepešní

Jelikož jsem na programu pracoval sám, tak jeho funkčnost, co se týče práce s formuláři, je o proti excelu pouze malá. Proto se naskytují další návrhy na možné vylepšení programu. Aplikace

prozatím umí jen jednoduchou sloupcovou operaci a to sčítání stejných hodnot, které uživatel zadá pomocí vyběrových tlačítek, které jsou volně definovatelné při tvorbě šablony formuláře. Proto jednou z navrhovaných možností rozšíření je více variabilní definice počítání hodnot z formulářů.

Další možnost rozšíření by byla interakce s jinými aplikacemi, které jsou již na trhu a jsou používány pro správu artefaktů při vývoji softwaru.

6 Závěr

Cílem této diplomové práce bylo seznámení se s problematikou provádění revizí při vývoji softwaru za dodržení standardu Automotive SPICE a zmapovat dostupný software, který je již na trhu a měl by tuto funkcionalitu integrovanou.

V teoretické části (kapitoly 2. a 3.) jsem rozebral standard Automotive SPICE vysvětlil jaké procesy v rámci životního cyklu využívá, jak se tyto procesy hodnotí, kolik existuje v rámci tohoto standardu úrovní procesu schopnosti v organizaci. Dále jsem se zaobíral konkrétními druhy revizí, které se provádějí. Vysvětlil jsem, jak celý proces tohoto revidování probíhá a jaké jednotlivé role mohou mít účastníci, kteří se podílí na revidování softwaru.

Následně jsem vybral komerční softwary, které jsem podrobil výzkumu za cílem zjištění, jestli mají podporu revidování jednotlivých artefaktů, které jsou součásti vývoje. Z tohoto výzkumu vyplynulo, že pouze polovina mnou testovaných softwaru má tuto funkcionalitu.

V poslední kapitole jsem se věnoval praktické části této diplomové práce. Kde cílem bylo vytvořit aplikaci, která by tuto funkcionalitu umožňovala a zároveň dokázala stahovat data (artefakty) z nástroje Rational DOORS Next Generation, který byl součásti výzkumu. A tyto data vkládat do vygenerovaných formulářů, které uživatel specifikuje. Aplikace, která vznikla je pouze prototyp a k jejímu reálnému použití by bylo třeba implementovat další funkcionality, které by umožňovaly širší možnost generování revizních formulářů a následné výpočetní funkcionality nad těmito formuláři.

Vzniklá aplikace slouží spíše jako rozšíření Rational DOORS Next Generation, proto je nemožné ji porovnat s aplikacemi, které jsem podrobil průzkumu. Avšak při tvorbě aplikace jsem se nechal inspirovat některými funkcionalitami, kterou nabízely aplikace, které byly součásti průzkumu.

Literatura

[1] Automotive SPICE® Process Reference and Assessment Model (PDF 1800KB) - RELEASE 3.1 - 01 November 2017, http://www.automotivespice.com/download/

[2] Sommerville, I. (2011). Software engineering. Boston: Pearson.

[3] Software Review. (2018, March 13). Retrieved January 8, 2019, from http://www.professionalqa.com/software-review

[4] Structured Walkthrough. (2018, January 16). Retrieved January 9, 2019, from http://www.professionalqa.com/structured-walkthrough

[5] Code Review. (2016, September 5). Retrieved January 11, 2019, from http://www.professionalqa.com/code-review

[6] S.r.o., E. (n.d.). Software Download. Retrieved February 18, 2019, from https://www.reqview.com/download.html

[7] IEEE Std. 1028-1997, IEEE Standard for Software Reviews

[8] Best Requirements Management Software | 2019 Reviews of the Most Popular Systems. (n.d.). Retrieved February 5, 2019, from https://www.capterra.com/requirements-management-software/

[9] FOWLER, Martin. Patterns of enterprise application architecture. Boston: Addison-Wesley, c2003. ISBN isbn9780321127426.

[10] Ardalis. (n.d.). Zobrazení v ASP.NET Core MVC. Retrieved April 17, 2019, from https://docs.microsoft.com/cs-cz/aspnet/core/mvc/views/overview?view=aspnetcore-2.2

[11] Rick-Anderson. (n.d.). Referenční příručka syntaxe Razor pro ASP.NET Core. Retrieved April 16, 2019, from https://docs.microsoft.com/cs-cz/aspnet/core/mvc/views/razor?view=aspnetcore-2.2

[12] Getting Started. (n.d.). Retrieved April 16, 2019, from https://developer.mozilla.org/en-US/docs/Web/Guide/AJAX/Getting_Started

[13] CHAPTER 5. (n.d.). Retrieved April 17, 2019, from https://www.ics.uci.edu/ fielding/-pubs/dissertation/rest_arch_style.htm

[14] Rowanmiller. (n.d.). Přehled – EF Core. Retrieved April 17, 2019, from https://docs.microsoft.com/cs-cz/ef/core/

[15] Rick-Anderson. (n.d.). Začínáme v ASP.NET Core – nová databáze – EF Core. Retrieved April 17, 2019, from https://docs.microsoft.com/cs-cz/ef/core/get-started/aspnetcore/new-db?tabs=visual-studio

[16] Rowanmiller. (n.d.). Začínáme v ASP.NET Core – existující databáze – EF Core. Retrieved April 17, 2019, from https://docs.microsoft.com/cs-cz/ef/core/get-started/aspnetcore/existing-db

[17] Auth0.com. (n.d.). JSON Web Tokens Introduction. Retrieved April 17, 2019, from https://jwt.io/introduction/

[18] (n.d.). Retrieved January 20, 2019, from https://jazz.net/wiki/bin/view/Main/RRCReportableRestAPI

[19] Why OSLC? (n.d.). Retrieved April 18, from https://open-services.net/why/

[20] Ruelas, G. (n.d.). Using OSLC capabilities in the Requirements Management application. Retrieved April 18, 2019, from https://jazz.net/library/article/1197

[21] Guardrex. (n.d.). Hostitele ASP.NET Core ve Windows se službou IIS. Retrieved April 26, 2019, from https://docs.microsoft.com/cs-cz/aspnet/core/host-and-deploy/iis/?view=aspnetcore-2.2#install-the-net-core-hosting-bundle

A Instalace a spuštění aplikace

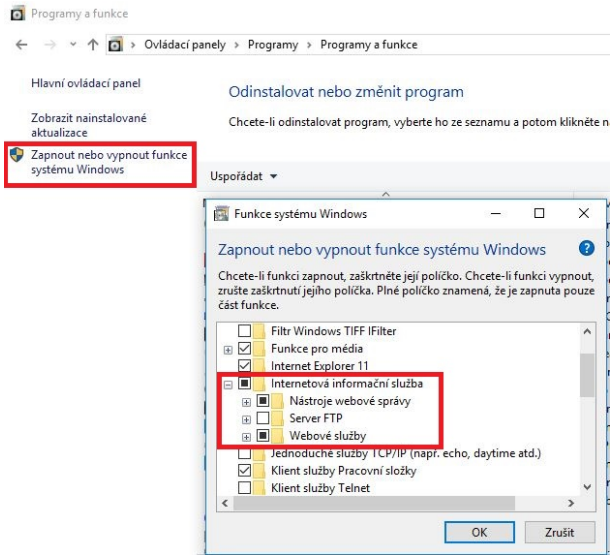
Spuštění celé aplikace je možno více způsoby jedním je otevřít zdrojové kódy ve vývojovém studiu v tomto případě Visual Studio 2017 Enterprise a druhou možností je nahrát soubory na lokální IIS.

Nutností je při spuštění jakýmkoliv způsobem zmíněným výše mít vytvořenou databázi a vložit connection string do souboru *appsettings.json*. Celý skript, který vytvoří databázi se nachází v souboru *DBinit.sql*, který obsahuje příkazy k vytvoření celé struktury databáze.

A.1 Instalace IIS

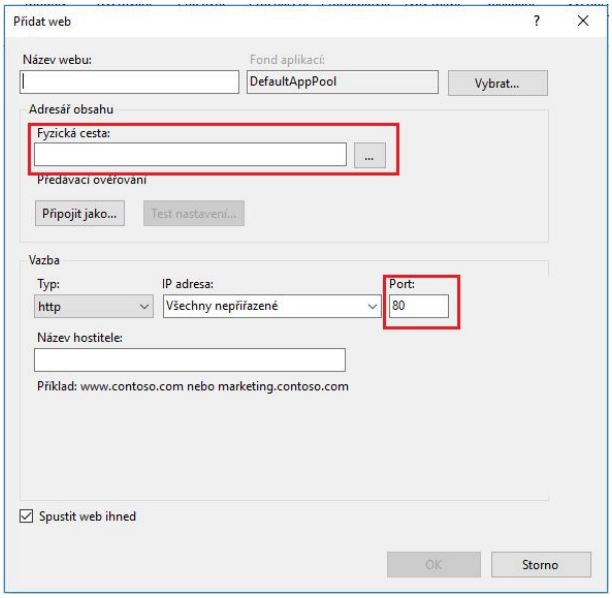
Aby bylo možné spustit aplikaci bez použití Visual Studia, je nutné mít operační systém Windows 10 a nainstalované IIS ne svém lokálním počítači. Dále je nutné nainstalovat balíček, který umožňuje vytvořit hosting pro ASP.NET Core aplikace IIS, který je dostupný zde [21].

Může se stát, že IIS není aktivní, proto je nutné tuto službu zapnou. Zapnutí se provede otevřením Ovládací panely > Programy > Programy a funkce a následně vybrat možnosti, které jsou na Obrázku 32. a potvrdit.



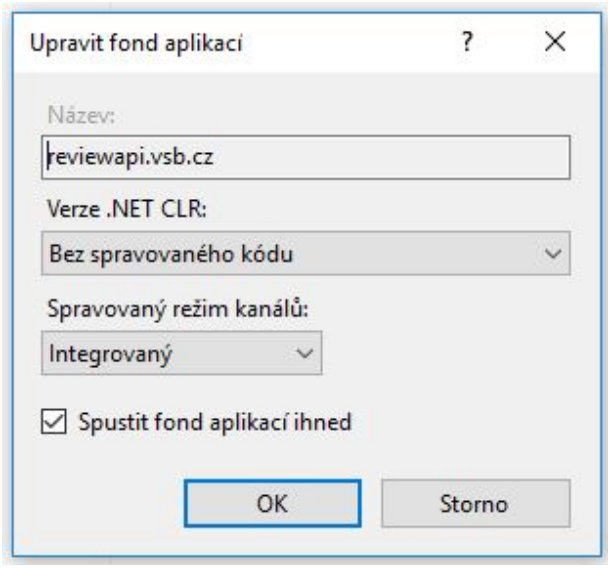
Obrázek 32: Spuštěni IIS

Následně je nutné vytvořit v IIS stránky, pod kterými budou části aplikace dostupné. Pro vyvolání konfigurace IIS Spustit>inetmgr a následně přidat web.



Obrázek 33: Vytvoření stránky

Fyzická cesta: zde je nutno nastavit cestu k adresáři APIrelease, který obsahuje veškeré soubory, které jsou nutné k nasazení aplikace. Číslo portu nastavit na 55188. Stejný postup se opakuje pro adresář WEBrelease, jen číslo portu je třeba nastavit na 49727. Oba adresáře naleznete v příloze práce. V záložce Fondy aplikací vyhledejte svou aplikaci a nastavte viz Obrázek 34.



Obrázek 34: Nastaveni aplikace

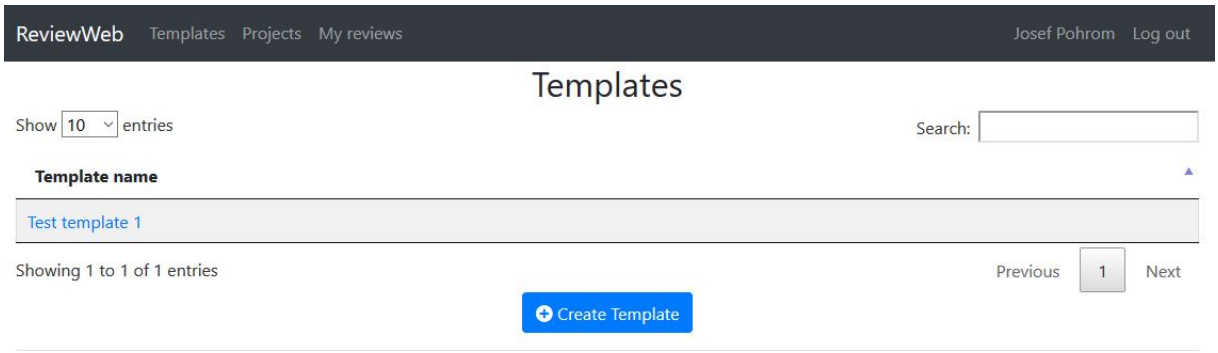
Toto proveďte pro obě aplikace (API, WEB). Nyní lze aplikace zapnout webová aplikace bude dostupná pod adresou localhost:49727.

B Uživatelský manuál

V této části je obsažen zkrácený uživatelský manuál, který ukáže jak vytvořit revizní formulář a následně založit revizi pro určité artefakty.

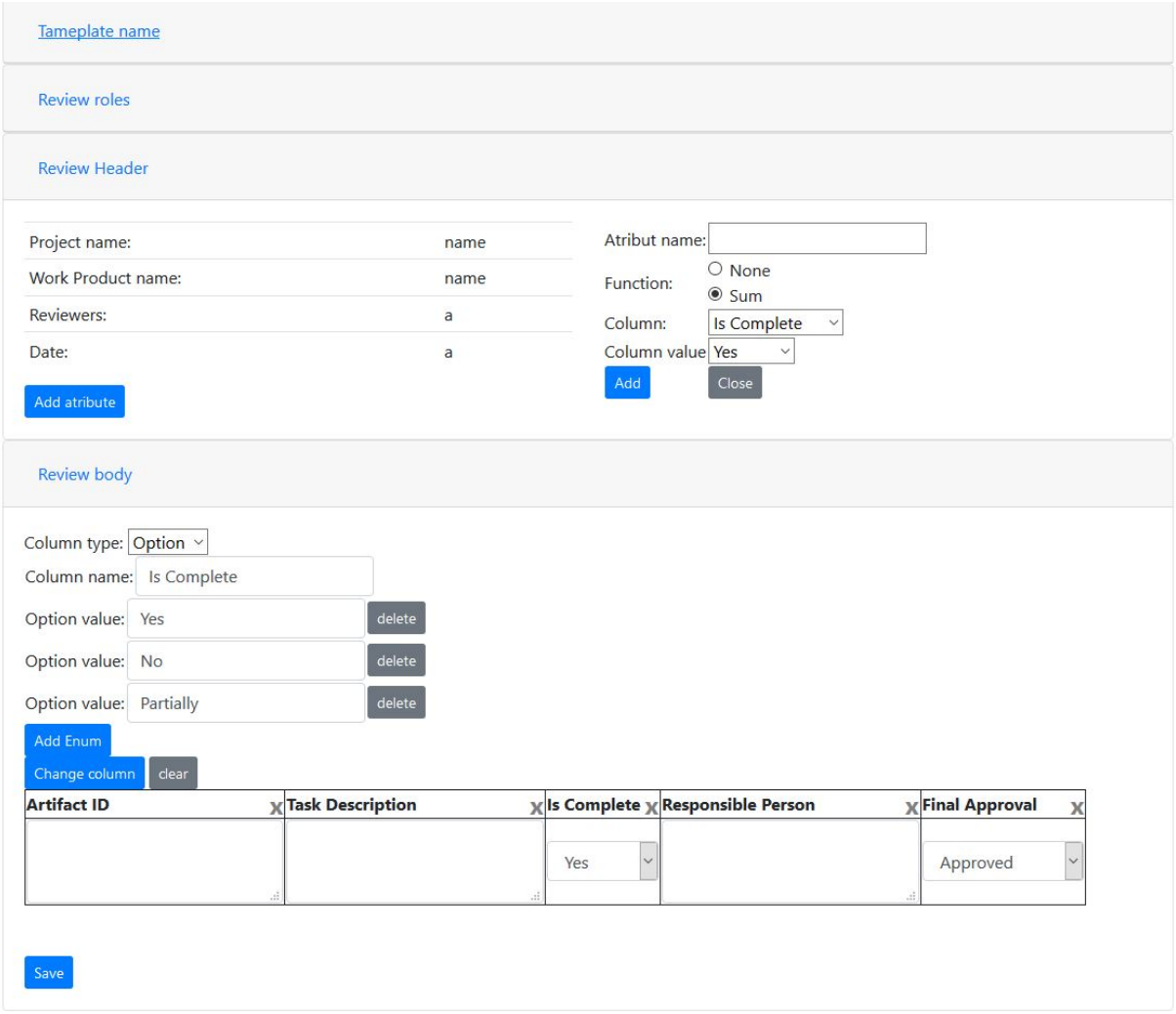
B.1 Vytvoření formuláře

Po přihlášení do aplikace klikněte na záložku *Templates*. Zobrazí se všechny dostupné šablony, které jsou definovány v aplikaci.



Obrázek 35: Všechny revizní šablony

Po kliknutí na tlačítko *Create Template* se zobrazí formulář pro definování revizní šablony, kterou následně lze využívat v aplikaci. Zde lze nastavit jednotlivé role, které lze následně uživatelům přiřazovat při vytváření revize. Následně lze definovat hlavičku, které bude součásti revize a tělo revizního formuláře. Vytvořit lze dva druhy formulářových vstupů (Text, Option) Při vytvoření Textového vstupu do formuláře se pouze zadává jméno sloupce. Při vytváření *Option* vstupu je nutné i definovat jednotlivé možnosti, které následně budou na výběr. Toto lze vidět na Obrázku 36



Obrázek 36: Definování šablony

Následně lze nastavit atributy do hlavičky revize, které jsou počítány jako například kolikrát je v sloupci *Is Complete* hodnota Yes viz Obrázek 36. Úprava sloupce, který byl náhodou definován špatně se spustí dvojitým poklepáním na hlavičku sloupce.

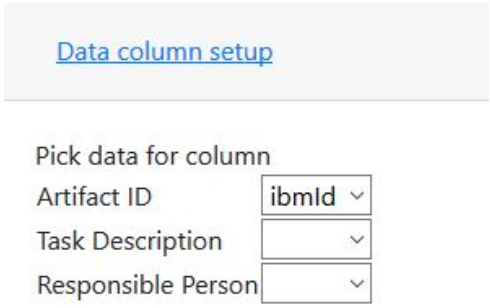
B.2 Vytvoření revize

Pro vytvoření revize je nutné vytvořit projekt a pracovní produkt(work product) uvnitř projektu. Toto se provádí v záložce *projects*. Po vytvoření projektu a pracovního produktu (work product) otevřete daný pracovní produkt a importujte data ze sestavy z aplikace Rational DOORS Next Generation. Klepnutím na tlačítko *Add Artifacts*.

Obrázek 37: Navigační tlačítka pracovního produktu

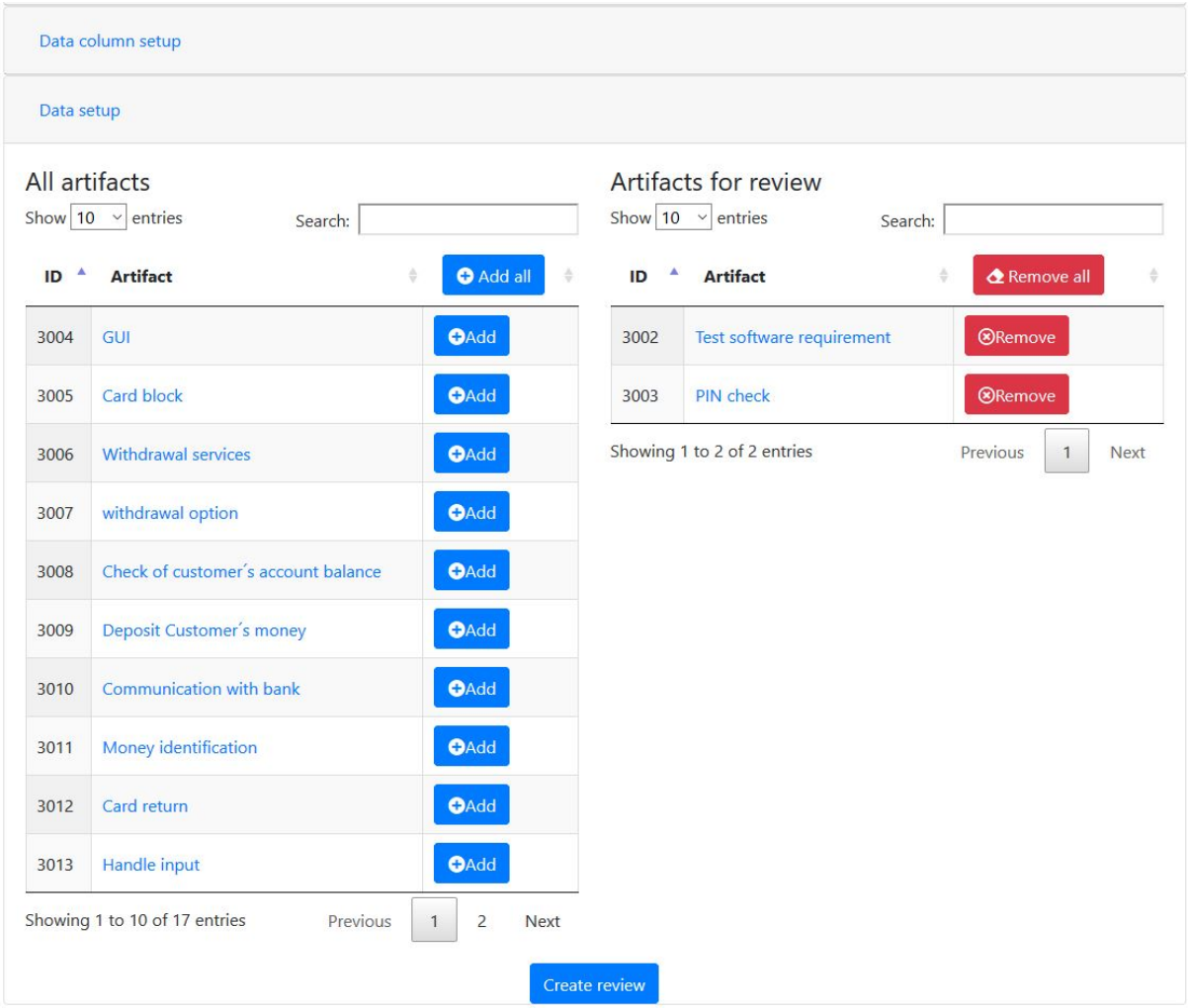
Následně je nutné vyplnit přihlašovací jméno a heslo do Rational DOORS Next Generation a zadat odkaz, na kterém se sestava s daty nachází. Následně se zobrazí všechny importované artefakty.

Po kliknutí na tlačítko *Start review* se zobrazí formulář pro definování revize v záložce *Review setup* se vyplňte všechna pole a vyberte šablonu formuláře, která bude použita pro danou revizi. Nastavte, kteří uživatelé se budou podílet na revizi. To se provede v záložce *User setup*. V záložce *Data column setup* se vybírá, do kterého sloupce se dané data zapíší. to lze vidět na Obrázku 38.



Obrázek 38: Přiřazení dát do sloupce

Následně je již potřeba pouze vybrat data, které budou zařazeny do revize. Tento výběr viz Obrázek 39.



Obrázek 39: Vybrání artefaktů, pro revizi

Po kliknutí na tlačítko Create review se stránka přesměruje na celý revizní formulář i s daty. Při vyplňování formuláře je vždy nutné, před ukončením zmáčknout tlačítko *save*, aby se uložily dané změny do databáze.